Titre: Title:	Semidefinite programming approaches and software tools for quadratic programs with linear complementarity constraints
Auteur: Author:	Patricia Lynn Gillett
Date:	2016
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Gillett, P. L. (2016). Semidefinite programming approaches and software tools for quadratic programs with linear complementarity constraints [Thèse de doctorat, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/2275/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/2275/
Directeurs de recherche: Advisors:	Miguel F. Anjos
Programme: Program:	Mathématiques de l'ingénieur

UNIVERSITÉ DE MONTRÉAL

SEMIDEFINITE PROGRAMMING APPROACHES AND SOFTWARE TOOLS FOR QUADRATIC PROGRAMS WITH LINEAR COMPLEMENTARITY CONSTRAINTS

PATRICIA LYNN GILLETT
DÉPARTEMENT DE MATHÉMATIQUES ET DE GÉNIE INDUSTRIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIÆ DOCTOR
(MATHÉMATIQUES DE L'INGÉNIEUR)
JUILLET 2016

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée :

SEMIDEFINITE PROGRAMMING APPROACHES AND SOFTWARE TOOLS FOR QUADRATIC PROGRAMS WITH LINEAR COMPLEMENTARITY CONSTRAINTS

présentée par : GILLETT Patricia Lynn
en vue de l'obtention du diplôme de : Philosophiæ Doctor
a été dûment acceptée par le jury d'examen constitué de :

M. AUDET Charles, Ph. D., président
M. ANJOS Miguel F., Ph. D., membre et directeur de recherche
M. GUÉRIN Jean, Ph. D., membre
Mme BEHJAT Laleh, Ph. D., membre externe

DEDICATION

To Junpei. I can't wait to see what happens next.

ACKNOWLEDGEMENTS

To Miguel Anjos, thank you for supporting me as I explored the relationship between theory and software. I finish my PhD well prepared for a future in industry because I was allowed the freedom to shape my research experience.

To Joaquim Judice, thank you for taking an interest and loaning us the benefit of your vast experience with complementarity problems.

To my friends at GERAD, thank you for the tea breaks, the long talks, and the whiteboard drawing sessions.

To my friends across the ocean in Yoshikawa Lab, thank you for showing me that research could be about making things.

To my friends in Montreal's tech community, thank you for opening up a world of possibilities to me.

To Junpei and my parents, thank you for always knowing how to make a few thousand kilometers feel like nothing.

I owe a great debt to the many open source developers whose work I have used, learned from, or expanded upon, including but not limited to:

- CVXOPT: Martin Andersen, Joachim Dahl, and Lieven Vandenberghe
- CVXPY: Steve Diamond and the CVXPY Github community
- QPECgen: Houyuan Jiang and Danny Ralph
- PyNEOS: Dominique Orban

I am also grateful to have had access to the NEOS server, a fantastic resource. The NEOS server is currently administered at the University of Wisconsin by Michael Ferris and Jeff Linderoth with the help of many other contributors. I give special thanks to Hans Mittleman and Rosemary Berger for their assistance.

RÉSUMÉ

Dans le domaine de la théorie des jeux, il est intéressant de créer un équilibre dynamique entre les agents afin qu'ils s'influencent de façon asymétrique. Le meneur affecte les règles du jeu, mais les choix subséquents du suiveur affectent la valeur de l'objectif du meneur. La dynamique meneur-suiveur est un outil puissant permettant de décrire un grand nombre de scénarios de jeux dans un contexte réel. Toutefois, les problèmes d'équilibre demeurent difficiles en pratique sauf pour quelques types de problèmes largement étudiés en théorie tel le problème linéaire bi-niveau.

Cette thèse tente de déterminer si les relaxations sous la forme de problèmes semi-définis, problèmes quadratiques avec contraintes de complémentarité linéaires sont efficaces. Cette classe de problème est équivalente aux problèmes d'équilibre. Une fonction objectif quadratique est particulièrement intéressante car la littérature dans ce domaine n'est pas complète et les relaxations semi-définies sont souvent efficaces pour les problèmes avec des fonctions objectif et/ou des contraintes quadratiques non-convexes.

Nous présentons une relaxation de base qui n'est pas coûteuse en temps de calcul puis nous discutons d'un grand nombre de contraintes qui permettent de resserrer la relaxation de façon significative. L'évaluation de l'efficacité de la relaxation, lorsque toutes ces contraintes sont utilisées, montre que cela mène à des difficultés d'implémentation numériques pour le solveur de points intérieurs qui résout le problème semi-défini. Nous discutons des raisons expliquant cela puis nous utilisons une autre approche afin d'éliminer cette difficulté. Cet algorithme démarre avec la relaxation de base renforcée avec une seule contrainte d'égalité agrégée puis ajoute de façon itérative des coupes resserrant la relaxation. Éventuellement, la relaxation est renforcée à son maximum alors que seule une fraction des coupes a été ajoutée. Les résultats numériques montrent que cette approche ne permet pas d'améliorer les bornes du problème semi-défini lorsque des coupes sont ajoutées. Ce n'est pas une faiblesse de la méthode mais cela démontre que le modèle de base est déjà une relaxation forte. Ainsi, l'aggrégation des contraintes en une seule contrainte d'égalité est très efficace pour renforcer la relaxation et ajoute peu de difficulté à son implémentation en pratique. Des recommandations sont émises concernant le choix des paramètres pour la méthode d'ajout de coupes de façon itérative.

Les relaxations semi-définies sont surtout utilisées pour borner les problèmes quadratiques difficiles. Les relaxations SDP des problèmes QPLCC peuvent être utilisées de cette façon, pour borner les noeuds des arbres branch and bound, mais nous sommes intéressés à utiliser toute l'information contenue dans la matrice de solution X^* du problème SDP. Lorsque cette matrice est de rang 1, elle peut être utilisée pour retrouver la solution globale dans l'espace d'état du problème original

non-relaxé. Nous définissons un point candidat comme étant un point estimant la solution globale d'un problème et nous présentons 4 façons de retrouver une solution dans l'espace d'état original pour une matrice X^* de rang arbitraire, X^* étant la solution de la relaxation SDP. Ce point candidat n'est pas spécifique aux problèmes QPLCC et pourrait être appliqué à d'autres problèmes. Des résultats numériques sont effectués afin de montrer que les points candidats sont des estimateurs de la solution globale. Nous présentons aussi des procédures afin d'utiliser les capacités de "warmstart" des solveurs en utilisant ce point candidat et démontrons leur impact.

En plus de contribuer à l'avancement des connaissances des problèmes QPLCC, nous avons aussi contribué à la communauté de recherche des logiciels traitant ces problèmes. Nous avons choisi Python comme langage de programmation puisque plusieurs librairies sont disponibles pour l'optimisation convexe, mais aussi pour sa capacité à interagir avec des solveurs externes codés dans d'autres langages de programmation. Nous avons créé des outils pour les problèmes QPLCC, par exemple en les formulant en langage AMPL et GAMS, nous avons résolu les QPLCC et/ou les problèmes SDP en utilisant des solveurs Pytyon, des solveurs installés localement ou la librairie NEOS. Tous les résultats numériques présentés dans cette thèse ont été effectués avec les librairies présentées dans cette thèse. Nous espérons que d'autres chercheurs dans le domaine QLPCC utiliserons nos librairies pour construire leurs propres méthodes de résolution et pour simplifier les comparaisons avec d'autres solveurs.

ABSTRACT

The leader follower dynamic seen in bilevel programming and equilibrium problems has the potential to unlock new doors in economic modeling and enable the realistic modeling of many problems of keen interest. The leader's choices affect the rules of the game which is played by the follower, but the follower's subsequent choices also impact the objective value achieved by the leader. Conceptually, the leader-follower dynamic is a valuable tool for describing any number of competitive real-world scenarios. However, to date equilibrium problems remain difficult in practice except for a handful of well studied problem classes such as the linear linear bilevel program.

This thesis is concerned with how effective semidefinite programming (SDP) relaxations can be constructed for quadratic programs with linear complementarity constraints (QPLCCs), a problem class which can equivalently model a class of equilibrium problems. The case of a general quadratic objective function is of particular interest since the literature in this area has not yet reached full maturity, and since semidefinite programming relaxations have often been effective for problems with nonconvex quadratic objective functions and constraints.

We present a base relaxation which is relatively computationally inexpensive, and then we present and discuss a number of tightening constraints which can have a dramatic tightening effect. However, in evaluating the effectiveness of the relaxation when all such constraints are used, we observe that blindly imposing all tightening constraints of the proposed types often leads to numerical difficulties for the interior point solver solving the semidefinite program. We discuss a possible reason for this, and finally we counteract it by developing an algorithm which begins with a middle ground model (the base relaxation strengthened with a single aggregated equality constraint) and iteratively adds tightening constraints to eventually obtain the tightness of the full relaxation while using a small fraction of the constraint pool in practice. In testing the iterative method with the middle ground model, we find that the SDP bound often doesn't improve over the course of the iterative method. This is not a flaw of the cut finding procedure, but instead demonstrates that the middle ground model is already as tight as the fully constrained model for these problems, establishing the aggregated equality constraint as an extremely effective strengthening measure which adds very little additional computational difficulty to the problem. Recommendations are made regarding parameter choice for the iterative method.

Semidefinite relaxations are most commonly used to bound difficult quadratic problems. SDP relaxations of QPLCCs can certainly be used this way, to bound nodes of a branch and bound tree, but we are also interested in using the information contained in the SDP solution matrix X^* to full advantage. SDP relaxations are commonly designed so that an SDP solution X^* which is rank one

can be mapped back to the space of the unrelaxed problem to give a global solution. We define the notion of a candidate point as a point which is intended to estimate a problem's global solution, and we present four ways a candidate point for an SDP relaxation solution X^* of arbitrary rank can be mapped back to the space of the original problem. The candidate point concept and definitions are not specific to the QPLCC and could be applied to other problems. We perform computational tests to support discussion of the different candidate points' suitability as an estimate of the problem's global solution. We also present procedures for assisting local or global solvers by warmstarting with the candidate point, and demonstrate the impact in both cases.

In addition to contributing to the state of knowledge for QPLCCs, it has also been our goal to contribute software to the research community working on these problems. We have chosen Python as our development language based on the existence of a number of good packages for numerical work generally and convex optimization specifically, and also based on its abilities to act as glue between other services such as external solvers in other languages. We have made tools for modeling QPLCCs, exporting them for other languages (AMPL, GAMS), formulating SDP relaxations, and solving QPLCCs and/or SDP problems using native Python solvers, locally installed languages/solvers, or the NEOS public server. All the computational work presented in this thesis has been executed using the packages presented in this paper. It is our hope that other researchers in the field of QPLCCs will use our packages to build their own solution methods and to simplify the process of testing against various solvers.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

BP	Bilevel Program
DSL	Domain Specific Language
EPEC	Equilibrium Problem with Equilibrium Constraints
FLOSS	Free, Libre, and Open Source Software
KKT	Karoush-Kuhn-Tucker
LCP	Linear Complementarity Problem
LLBP	Linear Linear Bilevel Program
LPCC	Linear Program with Complementarity Constraints
MPEC	Mathematical Program with Equilibrium Constraints
MPCC	Mathematical Program with Complementarity Constraints
OSS	Open Source Software
PSD	Positive Semidefinite
QBP	Quadratic Quadratic Bilevel Problem
QP	Quadratic Program
QPCC	Quadratic Program with Complementarity Constraints
QPEC	Quadratic Problem with Equilibrium Constraints
QPLCC	Quadratic Program with Linear Complementarity Constraints
SOC	Second Order Cone
SOC-RLT	Second Order Cone Relaxation Linearization Technique
SDP	Semidefinite Program, Semidefinite Programming
SQLP	Semidefinite, Quadratic, and Linear Programming

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

In modeling economic games, it is valuable to be able to model a leader-follower dynamic in which two parties influence one another in an asymmetric fashion. The leader's choices influence the rules of the game which is played by the follower, but the follower's subsequent choices also impact the objective value achieved by the leader. Consider for example the problem of an airline setting ticket prices. The airline is the leader player which aims to maximize its profit, while the customers are follower players who aim to meet their travel needs as cheaply as possible. To realize maximum profit, the airline must understand its customers' ticket buying strategies and seek a profitable equilibrium between itself and its customers.

In economics such leader-follower games are known as Stackelberg games (Stackelberg, 1952), and this two player dynamic is modeled in optimization using such problem classes as bilevel problems (BPs) and mathematical programs with equilibrium constraints (MPECs), which can under certain conditions be equivalently rewritten as mathematical programs with complementarity constraints (MPCCs). This thesis is concerned with developing solution methods for a subclass of MPCC known as the quadratic problem with linear complementarity constraints (QPLCC), which can model bilevel programs for which the upper and lower level objective functions are general quadratic and convex quadratic functions, respectively, and constraints are otherwise linear.

Bilevel and equilibrium dynamics intuitively describe many systems involving human behaviour and business strategy: businesses competing for market share, businesses choosing pricing strategies, governments setting policy, electricity providers bidding to provide energy to a power grid, etc. In each of these examples, a practical method to find improved solutions can easily lead to large scale profits. However, these problems are difficult to solve in practice and there is a tradeoff between models which are sophisticated enough to reflect a complex reality and models which are simple enough to be solved in practice. For the example of the power grid problem, the first hints of satisfactory tractability have begun to emerge in the last decade, see Bautista et al. (2007); Hu and Ralph (2007); Hu and Fukushima (2011); Dempe et al. (2015) for four takes on electricity networks from equilibrium perspectives. Alternatively, see Gabriel and Smeers (2006); Siddiqui and Gabriel (2013) for examples of natural gas market models. As research boundaries are pushed so that more complex classes of equilibrium problems can be practically solved, equilibrium problems stand to be a growing piece of the optimization landscape.

1.1 Basic definitions and concepts

1.1.1 Bilevel programming

What makes bilevel programming a powerful modeling tool is the ability to model situations (games) in which two parties have conflicting interests and there is an asymmetry in decision-making power. Opposing interests and leader-follower decision processes are both seen frequently in real-world games, so bilevel programming provides a much-needed extension to game modeling techniques which have important real-world applications in a number of fields. For example, the government takes the leader's role in the regulation of hazardous goods transportation by deciding which roads can or can't be used, keeping in mind that drivers will always choose to use the shortest path which is available to them. This behaviour can be seen again in oligopolistic utility markets such as electricity where typically a few firms lead and the rest of the market follows. For market pricing applications in particular, the difference between a global solution and a (non-global) local solution can represent a substantial difference in profits, making global solution methods highly desirable.

The general bilevel program takes the form

$$\begin{aligned} \min_{x\in\mathbb{R}^{n_x},\, y\in\mathbb{R}^{n_y}} \quad & f(x,y) \\ \text{s.t.} \quad & (x,y)\in X \\ & y\in \arg\min_{y'} \; F(x,y') \\ & \qquad \text{s.t.} \;\; y'\in\Omega_y(x) \;\; . \end{aligned} \tag{BP}$$

The variables of (BP) are partitioned into **upper level variables** x and **lower level variables** y.The constraint $(x,y)\in X$ is called an **upper level constraint**, and the problem $\min_{y'\in\Omega_y(x)} F(x,y')$ is called the **lower level problem** of (BP).

In the context of a Stackelberg game, the upper level variables are controlled directly by the leader while the lower level variables represent the follower's response. Note that the lower level problem is an optimization problem only in the variables y; upper level variables x are treated as fixed and known within the scope of the lower level problem. Furthermore, the lower level problem's objective function and feasible region depend on x.

Bilevel programs are classified according to the nature of each level's objective function and constraints. For example, a linear linear bilevel problem (LLBP) is a bilevel problem in which f and F are linear functions and X and $\Omega_y(x)$ are finite convex polytopes as in linear programming. In this

thesis we focus on QPLCCs of the type obtained by transforming linearly constrained QQBPs, i.e. the case where f and F are quadratic functions and X and $\Omega_y(x)$ are finite convex polytopes. We will further restrict ourselves to the case of convex quadratic F so that Karush-Kuhn-Tucker (KKT) conditions may be applied to the lower level problem to transform the QQBP to a QPLCC. However, we will allow f to be nonconvex as we are particularly interested in contributing to the state of knowledge for nonconvex QPLCCs.

We define a number of domains of interest in terms of (BP). The **relaxed feasible region** Ω is the set of (x, y) which satisfy the upper level constraints and lower level constraints, i.e.

$$\Omega = \{(x, y) \mid (x, y) \in X,\ y \in \Omega_y(x)\} \quad . \tag{1.1}$$

Unlike the admissible set explained below, points in the feasible set are not necessarily optimal for the lower level problem. For a given x, the set $\Omega_y(x)$ is the **feasible set of the lower level problem**.

The **trace** of the lower level problem with respect to the upper level variables is the set of x such that the lower level problem is feasible, i.e.

$$\Omega_x^2 = \{x \mid \Omega_y(x) \neq \emptyset\} \quad . \tag{1.2}$$

The **rational reaction set** $S(x)$ is defined as the set of y which optimally solve the lower level problem for a given x:

$$\begin{aligned} S(x) = \arg\min_{y'} \quad & F(x, y') \\ \text{s.t.} \quad & y' \in \Omega_y(x) \quad . \end{aligned} \tag{1.3}$$

The set of **admissible** (x, y), also known as the **induced region** or **inducible region**, is the set of solutions which are feasible and lie in $S(x)$, i.e.

$$\Upsilon = \{(x, y) \in X \mid y \in S(x)\} \quad . \tag{1.4}$$

Υ could be considered the **feasible region of the bilevel problem**, but that terminology is sometimes avoided to prevent confusion with the relaxed feasible region.

1.1.2 Complementarity problems

We define a general MPCC as having the form

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & z(x) \\ \text{s.t.} \quad & a_i(x) = 0 && \forall\ i \in E \\ & g_i(x) \leq 0 && \forall\ i \in I \\ & g_i(x)g_j(x) = 0 && \forall\ (i,j) \in C \quad . \end{aligned} \tag{MPCC}$$

For the purpose of this thesis, E and I are finite index sets corresponding to the sets of equality and inequality constraints, respectively.

The MPCC has decision variable vector x with dimension n, objective function $z : \mathbb{R}^n \to \mathbb{R}$, equality constraints given by $a_i : \mathbb{R}^n \to \mathbb{R}$ for $i \in E$, inequality constraints given by $g_i : \mathbb{R}^n \to \mathbb{R}$ for $i \in I$, and complementarity constraints $g_i(x)g_j(x) = 0$ which exist between inequality constraints i and j. The pairs of inequalities between which complementarity exists are given by C, with $C \subseteq \{(i,j) \mid i \in I,\ j \in I,\ i < j\}$.

This thesis is concerned with QPLCCs, which are MPCCs for which the objective function $z(x)$ is quadratic and $a_i(x)$, $g_i(x)$ are linear affine functions. Quadratic problems with complementarity constraints (QPCCs), and MPCCs in general, are a difficult class of problems. The Mangasarian-Fromovitz constraint qualification fails, the complementarity constraints are not differentiable, and the feasible region of the MPCC is not necessarily closed, convex, or connected (Mangasarian and Fromovitz, 1967; Luo et al., 1996).

We will now tie bilevel problems to mathematical problems with complementarity constraints. The general bilevel problem can be described more concretely as follows:

$$\begin{aligned} \max_{x,y} \quad & f(x,y) && && (1) \\ \text{s.t.} \quad & a_i(x,y) = 0 && i = 1...m_E^U && (2) \\ & g_i(x,y) \leq 0 && i = 1..m_I^U && (3) \\ & y \in \arg\min_{y'} \quad F(x,y') && && (4) \\ & \qquad \text{s.t.} \quad A_i(x,y') = 0 && i = 1...m_E^L && (5) \\ & \qquad\quad G_i(x,y') \leq 0 && i = 1...m_I^L \quad . && (6) \end{aligned} \tag{BP′}$$

Consider the lower level problem given by constraints (4)-(6) of (BP′). In the case where F, G_i, and A_i are all differentiable and some suitable constraint qualification is satisfied such as regularity, Slater's condition, or Mangasarian-Fromowitz, and the lower level objective function F is convex, the lower level problem may be equivalently replaced by its KKT conditions. That is to say, y is in the rational reaction set for the lower level problem if and only if there exist $\mu \in \mathbb{R}^{m_I^L}$ and $\lambda \in \mathbb{R}^{m_E^L}$ such that

$$\begin{gathered}\nabla_y F(x,y) + \sum_{i=1}^{p} \mu_i \nabla_y G_i(x,y) + \sum_{i=1}^{q} \lambda_i \nabla_y H_i(x,y) = 0 \ , \\ A_i(x,y) = 0 \qquad \forall\ i = 1...m_E^L \ , \\ 0 \le \mu \quad \perp \quad G(x,y) \ge 0 \ ,\end{gathered} \tag{1.5}$$

where $G(x,y) = \begin{bmatrix} G_1(x,y) \\ \vdots \\ G_{m_I^L}(x,y) \end{bmatrix}$.

The lower level problem of (BP′) can then be replaced by these equivalent conditions to yield an MPCC or alternatively an MPEC. MPECs are a class of problems which use a variational inequality to model the equilibrium state of some system. In the next subsection we will show the QPLCC which is obtained by performing this transformation on the QQBP discussed in the previous subsection.

1.1.3 Convexity, Cones, and Semidefinite Relaxations

> "In fact the great watershed in optimization isn't between linearity and nonlinearity, but convexity and nonconvexity."
>
> (*Rockafellar, 1993*)

A *convex function* $f(x)$ is one for which a line segment drawn between any two vectors $(x_1, f(x_1))$ and $(x_2, f(x_2))$ will not fall below the graph $y = f(x)$. Put in mathematical terms,

$$f(x) \le \theta f(x_1) + (1-\theta) f(x_2) \quad \forall\ x_1, x_2, \theta \in [0,1] \ , \tag{1.6}$$

or alternatively, a function f which is twice continuous and differentiable is convex if and only if its Hessian $\nabla_H(f)$ is positive semidefinite, which will be explained in more detail shortly.

A *convex set* C is one for which any convex combination of points in the set will also be in the set, i.e.

$$x_1 \in C, x_2 \in C \quad \Rightarrow \quad \theta x_1 + (1-\theta) x_2 \in C \quad \forall\ \theta \in [0,1] \ . \tag{1.7}$$

We will adopt the same definition of a convex optimization problem used by Boyd and Vanden-

berghe (2004). In standard form, a convex problem takes the form

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & g_i(x) \leq 0 \quad i = 1, \ldots, m \\ & a_i^T x = b_i \quad i = 1, \ldots, p \quad , \end{aligned} \tag{1.8}$$

where $f(x)$ and $g_i(x) \leq 0,\ i = 1, \ldots, m$ are convex functions. It is easy to show that the feasible region of (1.8) will be a convex set.

A convex programming class which is relied upon heavily in this work is conic programming. Conic programs take the form

$$\begin{aligned} \min_{x} \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & h - Gx \in \mathcal{K} \quad , \end{aligned} \tag{1.9}$$

where $\mathcal{K}$ is a closed convex cone. There are three cones of particular interest to us:

1. The non-negative orthant $\mathbb{R}^n_+$,
2. The second order cone SOC^{n+1}: $\left[\begin{smallmatrix} t \\ x \end{smallmatrix}\right]$ such that $\|x\|_2 \leq t^2$,
3. The positive semidefinite cone $\mathbb{S}^n_+$: x such that $vec(x)$ is a positive semidefinite matrix, where $vec : \mathbb{R}^{n^2} \rightarrow \mathbb{R}^{n \times n}$ is a function which reshapes a vector x into a square matrix following column major order.

A positive semidefinite (PSD) matrix is a Hermitian matrix X whose eigenvalues are all nonnegative. We denote the set of real $n \times n$ PSD matrices as $\mathbb{S}^n_+$, and we also interchangeably write $X \succeq 0$ to denote that X is PSD. $\mathbb{S}^n_+$ can equivalently be defined as the set of symmetric matrices $X \in \mathbb{R}^{n \times n}$ such that $z^T X z \geq 0 \ \ \forall\ z \in \mathbb{R}^n$.

It is worth noting that the former two cones can be expressed using semidefinite cone constraints, and also that multiple positive semidefinite constraints can be equivalently rewritten using a single larger semidefinite constraint, so without loss of generality a problem can be considered to be a semidefinite programming problem in the vein of (1.9) (with $\mathcal{K}$ replaced by $\mathbb{S}^m_+$ for some m) even if it contains a mixture of linear inequality constraints, second order cone (SOC) constraints, and positive semidefinite (PSD) constraints. Such problems can be solved using interior point methods in polynomial time (Nesterov et al., 1994).

1.1.4 Semidefinite programming relaxation technique

Throughout this thesis we discuss a number of semidefinite programming relaxations. All are constructed using the same basic technique, which we will introduce in this section. This technique can be applied to continuous optimization problems whose objective function and constraints can be expressed using quadratic functions. It has three main steps, Lifting, Relaxation, and Reinforcement, but first we introduce some notation.

Definition 1.1.1. *The dot operator '$\cdot$' denotes the inner product between two matrices having the same shape. For $A \in \mathbb{R}^{m\times n}, B \in \mathbb{R}^{m\times n}$,*

$$A \cdot B = trace(AB^T) = \sum_{i=1}^{m}\sum_{j=1}^{n} A_{ij}B_{ij} \quad . \tag{1.10}$$

For future convenience, we also build on this definition with the following lemma.

Lemma 1.1.2. *Given a matrix $M \in \mathbb{R}^{n\times n}$ and vector $y \in \mathbb{R}^{n\times 1}$, the scalar term $y^T My$ is equivalent to $M \cdot (yy^T)$.*

Proof. By expanding and refactoring $y^T My$ we find that

$$y^T My = \sum_{i=1}^{n}\sum_{j=1}^{n} M_{ij}y_i y_j = \sum_{i=1}^{n}\sum_{j=1}^{n} M_{ij}(yy^T)_{ij} \quad , \tag{1.11}$$

which by Definition 1.1.1 is equivalent to the dot product between M and the square matrix yy^T. □

We now proceed to explain the relaxation technique.

Lifting

Given a mathematical program (P) with continuous decision variables x, and assuming its objective function and constraints can be expressed using quadratic functions, we begin by defining a matrix X to be:

$$X = \begin{bmatrix}1\\ x\end{bmatrix}\begin{bmatrix}1 & x^T\end{bmatrix} = \begin{bmatrix}1 & x^T\\ x & xx^T\end{bmatrix} = \begin{bmatrix}1 & x_1 & x_2 & \cdots & x_n\\ x_1 & x_1^2 & x_1x_2 & \cdots & x_1x_n\\ x_2 & x_1x_2 & x_2^2 & \cdots & x_2x_n\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ x_n & x_1x_n & x_2x_n & \cdots & x_n^2\end{bmatrix} \quad . \tag{1.12}$$

Next, we create a new problem with constraint (1.12) and with the objective function and constraints of (P) rewritten as as a constraint and rewrite the objective and constraints in terms of the elements

of X, calling the new formulation (R). Quadratic terms in the elements of x can be written linearly in the elements of the matrix X.

For example, consider the problem given by (P^1_{Ex}), a very simple problem in two variables with a convex quadratic objective and a single nonconvex quadratic equality constraint:

$$\begin{array}{ll} \min\limits_{x\in\mathbb{R}^2} & x_2^2 \\ \text{s.t.} & ax_1^2 + bx_1x_2 + cx_2 = 3 \quad . \end{array} \tag{P^1_{Ex}}$$

The equivalent lifted problem with rank one constraint as in (1.12) is given by

$$\begin{array}{ll} \min\limits_{X} & X_{22} \\ \text{s.t.} & aX_{11} + bX_{12} + cX_{02} = 3 \\ & \text{rank}\left(\begin{bmatrix} 1 & X_{01} & X_{02} \\ X_{01} & X_{11} & X_{12} \\ X_{02} & X_{12} & X_{22} \end{bmatrix}\right) = 1 \quad . \end{array} \tag{R^1_{Ex}}$$

Problem (R^1_{Ex}) is equivalent to (P^1_{Ex}) and no more tractable. In (R^1_{Ex}), the difficulty of the problem has been captured by the last constraint which effectively enforces that X_{11} is equal to the square of X_{01}, X_{12} is equal to the product $X_{01}X_{02}$, and so on.

Relaxation

Since we cannot practicably enforce the rank one constraint, we will create the new problem (S) from (R) by replacing the rank one constraint on X with a semidefinite constraint $X \succeq 0$, which for such relaxations means X is in the positive semidefinite cone $\mathbb{S}^{n+1}_+$, where n is the number of variables in problem (P). Problem (S) is a relaxation of (R), and therefore a relaxation of (P), since

$$\{X \mid rank(X) = 1\} \quad \subset \quad \{X \mid X \succeq 0\} \quad . \tag{1.13}$$

Reinforcement

In this step, we can optionally derive additional constraints to impose on (S). Reinforcement is done by deriving valid quadratic constraints for (P), expressing them in terms of X, and adding them to (S). The goal is to tighten the feasible region of (S) to reflect that of (R) as closely as possible, so constraints added should be redundant in (R) but not (S). In particular, when dealing with nonconvex quadratic objective functions and a potentially unbounded relaxation, it is often important to impose constraints which directly or indirectly bound the diagonal elements of X.

1.2 Description of the problem

As previously mentioned, our motivation is to contribute to the state of knowledge for quadratic quadratic bilevel problems (QBPs) and QPLCCs, particularly those with nonconvex objective functions. In the QBP case, we restrict ourselves to problems with convex quadratic lower level objective functions. This restriction allows QBPs to be equivalently modeled as QPLCCs, as we will soon show. The QBP is defined as:

$$\begin{aligned}
\max_{x,y} \quad & \left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]^T Q_u \left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right] + p_u^T \left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right] + r \\
\text{s.t.} \quad & A_u \left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right] = b_u \\
& G_u \left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right] \leq h_u \\
& y \in \arg\min_{y'} \quad \left[\begin{smallmatrix} x \\ y' \end{smallmatrix}\right]^T Q_\ell \left[\begin{smallmatrix} x \\ y' \end{smallmatrix}\right] + p_\ell^T \left[\begin{smallmatrix} x \\ y' \end{smallmatrix}\right] \\
& \qquad \text{s.t.} \quad A_\ell \left[\begin{smallmatrix} x \\ y' \end{smallmatrix}\right] = b_\ell \\
& \qquad\qquad G_\ell \left[\begin{smallmatrix} x \\ y' \end{smallmatrix}\right] \leq h_\ell \quad .
\end{aligned} \tag{QQBP}$$

Since x is considered a constant within the lower level problem, without loss of generality the lower level objective function can be replaced by $y'^T Q_\ell^{yy} y' + x^T Q_\ell^{xy} y' + p_\ell^{yT} y'$. We are assuming convexity of the lower level problem, i.e. $Q_\ell^{yy} \succeq 0$, since the KKT conditions are both necessary and sufficient for linearly constrained problems with convex objective functions (Freund, 2004).

The KKT form of (QQBP) is the QPLCC:

$$\begin{aligned}
\max_{x,y} \quad & \left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]^T Q_u \left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right] + p_u^T \left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right] + r \\
\text{s.t.} \quad & A_u \left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right] = b_u \\
& G_u \left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right] \leq h_u \\
& A_\ell \left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right] = b_\ell \\
& G_\ell \left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right] \leq h_\ell \\
& \mu \geq 0 \\
& \mu_i (h_\ell - G_\ell \left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right])_i = 0 \qquad \forall\ i = 1 \dots p \\
& Q_\ell^{xyT} x + 2 Q_\ell^{yy} y + \sum_{i=1}^{p} \mu_i [G_\ell]_{i,n_x+1:}^T + \sum_{i=1}^{q} \lambda_i [H_\ell]_{i,n_x+1:}^T = -p_\ell^y \quad .
\end{aligned} \tag{QPLCC$_{\text{KKT}}$}$$

More generally, the problem we will develop models for is

$$
\begin{array}{lll}
\min\limits_{x} & z(x) = x^T Q x + p^T x + r & \\
\text{s.t.} & a_i^T x = b_i & \forall\ i \in E \\
& g_i^T x \leq h_i & \forall\ i \in I \\
& (h_i - g_i^T x)(h_j - g_j^T x) = 0 & \forall\ (i,j) \in C \quad .
\end{array}
\tag{P}
$$

where $x \in \mathbb{R}^n$. $z(x)$ is a quadratic function. Let $I = \{1, \ldots, m_I\}$ be the index set for inequality constraints and let $E = \{1, \ldots, m_E\}$ be the index set for equality constraints. Let $C \subseteq \{(i,j) \mid i \in I,\ j \in I,\ i < j\}$ be the set of pairs of indices (i,j) such that we have complementarity between the i^{th} and j^{th} inequality constraints.

It is important to note that QPLCCs are nonconvex optimization problems regardless of their objective functions, due to the disjunctive nature of complementarity constraints. For the complementarity constraint $(h_i - g_i^T x)(h_j - g_j^T x) = 0$, all solutions must satisfy at least one of $g_i^T x = h_i$ or $g_j^T x = h_j$. Each complementarity constraint presents a choice which may be branched upon, and it is possible (if impractical) to express the feasible region of the QPLCC as the union of the feasible regions of $2^{|C|}$ subproblems, where $|C|$ is the number of complementarity constraints in the QPLCC and where each subproblem corresponds to a different way that the complementarity constraints may be fixed.

However, in this thesis the term 'convex QPLCC' will be specially defined to indicate a QPLCC whose quadratic objective function is convex. Specifically, the objective function $f(x) = x^T Q x + p^T x + r$ is convex if and only if Q is positive semidefinite. Otherwise $f(x)$ is a nonconvex function and we say that it is a nonconvex QPCC. The techniques we propose are valid for QPCCs of both types.

As we will see later in the literature review, there exists equivalence between MIP, LLBP, and bilinear disjoint programs (BDPs), and one can be reformulated to another in polynomial time. This has implications for the complexity theory regarding these problem classes. For example, if there existed a polynomial time algorithm which could check the global optimality of a MIP solution, it would also be possible to check global optimality of LLBP solutions in polynomial time: the LLBP and its solution could be transformed to a MIP context and then the algorithm for MIPs could be applied, both polynomial time tasks. In point of fact, MIP, LLBP, and BDP are all NP-Hard because it is known that mixed integer programming is NP-Hard.

Nonconvex QPLCCs are NP-Hard in two regards: the problem of a linearly constrained nonconvex quadratic program is NP-Hard, and the objectiveless linear complementarity problem (LCP) is known to be NP-Hard as well. An important implication of nonconvex QPs being NP-Hard is that even if all complementarity constraints are branched on or relaxed, the remaining linearly

constrained nonconvex QP remains NP-Hard.

1.3 Research objectives

The first objective of this thesis is to investigate how effective SDP relaxations of QPCCs can be constructed. A number of tightening cuts are investigated and recommendations are made for two cases: the case where a single relaxation is heuristically constructed and solved, and the case where a cutting plane method is used to iteratively tighten the relaxation until the ideal relaxation is constructed.

The second objective of this thesis is to explore how a strong SDP relaxation can be used other than for its bounds. For a certain semidefinite relaxation technique, an SDP relaxation solution which has rank one can be mapped back to the space of the original problem to obtain a global solution. Motivated by the observation that this rank one condition rarely happens in practice even if the SDP relaxation is exact, we aim to develop an analogous derivation procedure which can be used to produce a candidate point from SDP relaxation solutions not having rank one. Such a candidate point can not be expected to be optimal or even necessarily feasible for the original problem, but provides a heuristic 'guess' which in practice is often very close to a globally optimal solution.

The final mission of this thesis is to produce open source software to help researchers working in similar areas. Computational research in the field of complementarity problems is historically fractured across many programming languages. This is natural since researchers approach MPCCs with techniques from a wide variety of fields, and each typically chooses the programming language best suited to their methods. However, this can make comparison with other solvers or methods excessively difficult. The software produced during this research program aims to provide a Python framework from which various tools from other languages can be accessed.

1.4 Structure of the thesis

This thesis is structured as follows. Chapter 2 presents a review of literature relevant to the work presented in this thesis. Chapter 3 presents a semidefinite relaxation formulation for QPLCCs and investigates the effectiveness of a number of tightening measures. A few relaxation formulations are evaluated and finally an iterative technique is presented. Chapter 4 considers how SDP relaxations can be used not just for their bounds but also to obtain a heuristic estimate of the global optimal solution which can be used to warmstart local and global NLP solvers. Chapter 5 introduces three software products developed during the research program: PyQPCC, PyQPECgen, and PySDPT3glue. Chapter 6 provides conclusions and summarizing discussion. Appendices A and B include full tables for an analysis performed in chapter 4.

CHAPTER 2 CRITICAL LITERATURE REVIEW

2.1 Game theory, bilevel programs, equilibrium problems, and complementarity problems

Nash equilibrium

The notion of equilibria in game theory originated with Nash's non-cooperative principle (Nash, 1951). The principle states that in a game with no collusion between players, an equilibrium state consists of a strategy for each player such that no player has an incentive to change their strategy unilaterally. That is to say, each player i's strategy minimizes their individual cost function under the assumption that the strategies of all other players $j \neq i$ are known and will not change.

Mathematically, let $Y_i \subseteq \mathbb{R}^{m_i}$ be the set of valid strategies for player i, for players $1 \dots M$. When other players' strategies are given by $y_{\neq i}^{\text{given}}$, the cost function for player i is denoted $\theta_i(y_i, y_{\neq i}^{\text{given}})$. In terms of this notation, Nash equilibrium is defined as a vector $y^* \in \prod_{i=1}^{M} Y_i$ which gives a strategy for each player such that

$$y_i^* \in \arg\min_{y_i}\{\theta_i(y_i, y_{\neq i}^*) : y_i \in Y_i\} \qquad \forall i = 1...M \quad . \tag{2.1}$$

Nash equilibrium does not imply solution quality, only its stability under non-cooperation. Even a Pareto inferior point can be a Nash equilibrium if no player can improve their objective unilaterally.

Stackelberg games

By contrast, the Stackelberg game presents a model for asymmetric games with a leader and one or more followers (Stackelberg, 1952). The leader's strategy x may be freely chosen from the set X, while follower i's feasible strategy set Y_i depends on x. In the multi-follower case, the set of feasible strategies for follower i is not affected by the choices other followers make, but the follower i's objective function may be. Then for a fixed leader strategy $x^{\text{given}} \in X$, the followers responses $y_i^*, i = 1...M$ must be in equilibrium with one another in the following sense:

$$y_i^* \in \arg\min_{y_i}\{\theta_i(x^{\text{given}}, y_i, y_{\neq}^*) : y_i \in Y_i(x^{\text{given}})\} \qquad \forall i = 1...M \quad . \tag{2.2}$$

Nash games and Stackelberg games can also be generalized yet a step further with equilibrium problems with equilibrium constraints (EPECs) and multi-leader-follower games, as in Hu and Ralph (2007) and Siddiqui and Gabriel (2013).

Origin of bilevel programming

Beginning in 1973, Bracken and McGill proposed the use of bilevel programs (then called *mathematical programs with optimization problems in the constraints*) as a modeling tool (Bracken and McGill, 1973) and published papers giving bilevel programming models for applications in a number of fields. The most well known of these applications is a problem of choosing production and marketing levels to secure the desired market share in a market with competition (Bracken and McGill, 1978).

Optimistic vs. pessimistic formulations

As discussed in section 1.1, bilevel programming is founded on the premise that the leader can correctly predict how the follower will react. However, the follower's problem may have multiple optimal solutions which are equally attractive to the follower but not necessarily so for the leader. The bilevel programming models seen throughout this thesis are formulated according to the optimistic assumption that the follower will 'break ties' among solutions in the rational reaction set $S(x)$ (as defined in equation 1.3) by choosing the solution which benefits the leader the most.

By contrast, the pessimistic formulation is posed as

$$\begin{array}{lll} \max\limits_{x,y} & f(x,y) & \\ \text{s.t.} & (x,y) \in X & \\ & y \in \arg\min\limits_{y} & f(x,y) \\ & & \text{s.t.} \quad y \in S(x) \quad . \end{array} \tag{BP$_{PESS}$}$$

Under the pessimistic assumption, the leader assumes that the follower will 'break ties' by choosing the strategy which least benefits the leader. Although the pessimistic model is sometimes presented as a maxmin problem, the (BP$_{PESS}$) formulation is more correct because the leader is forced to choose a value for x which makes the lower level problem feasible (Audet et al., 1999). The leader therefore chooses its strategy in order to protect against the worst case. Although there is literature on both models, the optimistic model is by far the more frequently studied as the pessimistic formulation involves an additional layer of complexity.

Complementarity problems as MIP$_{0-1}$

In addition to complementarity problems, bilevel problems are also closely related to mixed 0-1 integer programs (MIP$_{0-1}$).

Consider a constraint of the form $0 \leq h_0 - g_0^T x \perp h_1 - g_1^T x \geq 0$.
If constants M_0, M_1 are known such that for all feasible x,

$$0 \leq h_0 - g_0^T x \leq M_0 \qquad \text{and} \qquad 0 \leq h_1 - g_1^T x \leq M_1 \ , \tag{2.3}$$

then the original complementarity constraint can be rewritten linearly with the use of a binary variable z as follows:

$$\begin{aligned} &0 \leq h_0 - g_0^T x \leq M_0 z \\ &0 \leq h_1 - g_1^T x \leq M_1 (1 - z) \\ &z \in \{0, 1\} \ . \end{aligned} \tag{2.4}$$

Constraints which use a large constant and a binary variable to control whether or not an expression is forced to zero are known as big-M constraints. When used to model a complementarity, the binary variable z directly corresponds to choosing which expression to force to zero. If appropriate constants M exist we can reformulate all complementarities this way, resulting in a MIP_{0-1} with $|C|$ binary variables, where C is the set of complementarity constraints.

In practice, constants M are not in fact required since the complementarities can be branched on implicitly: for example, the constraint $h_0 - g_0^T x \perp h_1 - g_1^T x$ can be branched on by creating two subproblems, one where the complementarity is replaced by the equality constraint $h_0 - g_0^T x = 0$, and the other where it's replaced by $h_1 - g_1^T x = 0$.

Audet et al. (1997) showed how a MIP_{0-1} may be formulated as an LLBP, and also showed equivalences between algorithms for MIP_{0-1} and LLBP. The significance of this result is that any algorithmic improvements for one can be extended to the other as well. This is further explored by Audet et al. (2007a) through the development of an algorithm for LLBP which is analogous to a branch and cut algorithm for MIP_{0-1}, and by Audet et al. (2007b) through the application of disjunctive programming results to derive valid cuts which tighten the linear relaxation of the bilevel problem.

2.1.1 The challenge of globally solving BPs and MPCCs

Bilevel programs are difficult to solve globally; globally solving even the relatively simple LLBP was shown by Hansen et al. (1992) to be strongly NP-Hard, and Vicente et al. (1994) showed that checking local optimality is also strongly NP-Hard. Furthermore, in general the complementarity structure hidden within bilevel programs renders the admissible region nonconvex and not necessarily closed or connected (Luo et al., 1996).

In the case where a bilevel problem's lower level problem has a convex objective function, both objective and constraints are differentiable, and an appropriate constraint qualification is satisfied,

the lower level problem can be replaced by its KKT conditions to yield an equivalent single level program.

When the KKT form of an LLBP is found, the resulting problem is a linear program with complementarity constraints (LPCC). The literature for the LPCC and its objectiveless cousin the LCP is by far the most mature among MPCC problem classes. For these problem classes, approaches range from active set methods to penalty methods and beyond. Extensive research has been done on global solution methods for LPCCs, such as Hansen et al. (1992); Hu et al. (2008); Bai et al. (2013).

In addition to LPCCs, the case of convex QPLCCs has also been well studied, often with the use of semidefinite programming techniques such as in Bai et al. (2013); Braun and Mitchell (2005). The convex QPCC shares an important property with the LPCC, which is that it can be globally solved by enumerating and solving the $2^{|C|}$ convex subproblems which correspond to all possible ways that the complementarity constraints could be satisfied. Each subproblem is a linearly constrained convex QP and can be solved with relative ease. QPLCCs with non-convex objectives, however, remain challenging in practice, especially when a global solution is desired.

2.2 Non-exhaustive review of solution methods for BPs and MPCCs

Active set methods

Active set methods are a broad class of methods for solving complementarity problems. For a problem with m complementarity constraints, a binary vector $\alpha \in \mathbb{B}^m$ is used as a tool to represent a particular complementarity assignment. Each complementarity assignment is associated with a subproblem in which each complementarity constraint $f(x)g(x) = 0$ has been replaced with either $f(x) = 0$ or $g(x) = 0$. The solution to the original complementarity problem can be thought of as the best solution found among all 2^m such subproblems, and techniques such as branch-and-bound can be applied.

The umbrella of active set methods includes not just versions of integer programming standards such as branch-and-bound and branch-and-cut (ex. Hansen et al. (1992) for LLBP), but also any other methods that focus on identifying the binary ‘complementarity alignment’ associated with an optimal solution. For example, there is a method by Hu et al. (2012) which formulates an LPCC as a minimax integer program using the active set formulation, then solves the problem using a Logical Bender’s approach. More recently, Bai et al. (2013) extended this work to the case with a convex quadratic objective function.

Branch and bound methods (MPCCs)

Branch and bound approaches branch on the choice inherent in each complementarity. This can be done explicitly by rewriting each complementarity using two big-M constraints and a binary variable to be branched on, but if M is not known the constraint can still be branched on implicitly by forming two subproblems: to branch on the complementarity constraint $x_1 x_2 = 0$, replace it in one subproblem by the linear constraint $x_1 = 0$, and in the other subproblem by $x_2 = 0$.

Interior point methods (MPCCs)

KNITRO, a nonlinear program (NLP) solver which can handle complementarity constraints, does so by applying an interior point method with an l_1 penalty term to the objective. However, it may be difficult to infer which constraints are active in the optimal solution, so the solution is also post-processed by applying a user-determined number of iterations of an active set method to try and determine a more exact solution. This method is shown by Leyffer et al. (2006) to converge globally to a strongly stationary point.

In recent years Coulibaly and Orban (2012) also developed an l_1 elastic interior point method, with the advantage that the modified problem is known to have an interior and satisfy the Mangasarian-Fromowitz constraint qualification. This method is particularly well suited to degenerate problems.

Convexification (MPCCs)

MPCCs can be viewed as general NLPs where the complementarity constraints become nonconvex quadratic equality constraints. Some additional handling is needed in order to solve MPCCs as NLPs, such as that seen in Bautista et al. (2007); Nguyen et al. (2011). Bautista et al showed how, for a particular MPEC, the complementarity constraints of the corresponding NLP can be convexified by the use of a smoothing term and then solved using commercial NLP software.

Parametric complementary pivot (MPCCs)

This method operates by progressively lowering a parameter z until no feasible solution can be found with objective value lower than it. Initially z is set to $+\infty$. A feasible point (x, y) is found such that $F(x, y) \leq z$. z is updated by setting $z = F(x, y)$ and the constraint $\nabla_y L = 0$ is perturbed in order to insure that the same (x, y) isn't returned again but the optimal solution is not changed. This procedure is repeated until it is no longer possible to find (x, y) such that $F(x, y) \leq z$. It has been shown by Colson et al. (2005) that this method does not necessarily converge to the optimal solution.

Extreme point methods (LLBPs)

These methods are based on the idea of enumerating vertices by exploring bases for the lower level problem. Extreme point methods are made possible by the fact that if an LLBP has a solution, the solution set is guaranteed to contain at least one extreme point of the relaxed feasible region Ω.

Descent methods (LLBPs)

Descent methods require that the solution to the lower level problem is unique for every x, and we can we can write y as $y(x)$. We then consider the problem of looking for a 'rational descent direction', i.e. a direction d and step size α s.t. $x + \alpha d$ ($\alpha > 0$) stays rational for the bilevel problem while giving a sufficient decrease in $F(x, y(x))$.

Methods vary widely depending on the nature of the objective functions and constraints used, ranging from problems with no upper level constraints to problems where both objectives are convex quadratic and all other constraints are linear (Colson et al., 2005; Vicente et al., 1994).

Penalty function methods (LLBPs)

These methods suggest replacing the lower level problem $\min_y f(x, y)$ s.t. $g(x, y) \leq 0$ with the unconstrained problem $\min_y f(x, y) + r\phi(g(x, y))$, where r is a positive scalar and $\phi(\cdot)$ is a barrier-type function which is always positive and approaches $+\infty$ as y approaches the boundary of $\{y : g(x, y) \geq 0\}$ (Colson et al., 2005).

2.3 Literature applying SDP to MPCCs and related problems

SDP relaxations in branch and bound

There is precedent for the use of SDP rather than LP relaxations in a branching scheme, meaning that SDP relaxations are solved at nodes of a branching tree. For example, Armbruster et al. (2008) implemented a branch and cut method for the minimum graph bisection problem, solving SDP relaxations at each node. In general, an SDP relaxation is expected to take longer to solve than an LP relaxation at a given node, but if the SDP relaxation gives bounds that are significantly tighter then more pruning can occur in the tree so that only a small number of nodes are evaluated. Potentially, the reduction in the size of the tree can compensate for the increased computational complexity, resulting in a more efficient method overall.

An exact SDP relaxation for a special nonconvex QPLCC

We will discuss the work of Ye and Zhang (2003) in some detail as it directly influences our work. Consider the following problem, a ball-constrained QPLCC:

$$\begin{array}{lll} \max\limits_{x} & x^T Q x + p^T x + r & \\ \text{s.t.} & \|x\|^2 \leq 1 & \\ & g_i^T x \leq h_i & \forall\ i \in I \\ & (h_i - g_i^T x)(h_j - g_j^T x) = 0 & \forall\ i \in I, j \in I, i < j \quad . \end{array} \tag{P_{YZ}}$$

The complementarity constraints of this model have a very specific structure, implying that for each pair of linear inequalities, at least one must hold with equality. This effectively means that at most one of the inequalities of I may be non-active. This is an example of an SOS1 constrained problem in disjunctive constraint modeling (Vielma and Nemhauser, 2011). SOS1 constraints are disjunctive constraints over continuous nonnegative variables in which at most one variable is allowed to be non-zero.

Ye and Zhang construct a relaxation of the type introduced in section 1.1.4:

$$\begin{array}{lll} \max\limits_{X} & \begin{bmatrix} r & \frac{p^T}{2} \\ \frac{p}{2} & Q \end{bmatrix} \cdot X & \\ \text{s.t.} & \sum\limits_{i=1}^{n} X_{ii} \leq 1 & \\ & X\bar{g}_i \in SOC & \forall\ i \in I \\ & \bar{g}_i^T X \bar{g}_j = 0 & \forall\ i \in I, j \in I, i \neq j \\ & X \geq 0, \quad X_{00} = 1 \quad , & \end{array} \tag{S_{YZ}}$$

where $\bar{g}_i = \left[\begin{smallmatrix} h_i \\ -g_i \end{smallmatrix}\right]$.

The reinforcing constraint $X\bar{g}_i \in SOC$ is derived from the observation that $\left[\begin{smallmatrix} 1 \\ x \end{smallmatrix}\right]$ is in the second order cone, which is closed under multiplication by nonnegative scalars such as $h_i - g_i^T x$. This construction of a new constraint from a second order cone constraint and a linear inequality constraint has been termed an SOC-RLT constraint by Burer and Anstreicher (2013), due to its being in the spirit of the reformulation-linearization technique introduced by Sherali and Adams (1999).

Two things are remarkable about relaxation (S_{YZ}). First, it is exact, meaning that it has the same optimal value as (P_{YZ}), even when Q is not positive semidefinite. Second, Ye and Zhang also present a polynomial time algorithm which, given an optimal solution X for (S_{YZ}) which is not necessarily rank one, computes a vector x which is optimal for (P_{YZ}). The algorithm relies on both the special complementarity structure and the unit ball constraint.

An SDP driven heuristic

For a convex QPLCC, Braun and Mitchell (2005) present a method which uses an SDP relaxation as a key component in a heuristic. A convex QPLCC with $|C|$ complementarity constraints can be solved by enumerating over a tree of $2^{|C|}$ linearly constrained convex quadratic subproblems. The proposed method solves an SDP relaxation of the QPLCC, then applies spectral decomposition to the optimal solution X^* to find its closest rank-one approximation $\lambda_1 \xi_1 \xi_1^T$. Information about ξ is then used in an attempt to deduce which way some complementarity contraints should be fixed. Those complementarity constraints which can be deduced with least confidence are retained while the rest are fixed, and the smaller enumeration tree for this greatly simplified convex QPLCC is explored one leaf node at a time. The heuristic presented is specific to QPLCCs with convex objective functions, but this paper spurs our interest in using the values of the SDP solution matrix heuristically rather than using SDP relaxations solely for their bounds.

CHAPTER 3 SEMIDEFINITE RELAXATIONS OF QUADRATIC PROGRAMS WITH LINEAR COMPLEMENTARITY CONSTRAINTS

The focus of this chapter is the construction of SDP relaxations for QPLCCs of the form

$$
\begin{array}{lll}
\min\limits_{x\in\mathbb{R}^n} & x^TQx + p^Tx + r & \\
\text{s.t.} & a_i^Tx = b_i & \forall\ i \in E \qquad (1)\\
& g_i^Tx \le h_i & \forall\ i \in I \qquad (2)\\
& (h_i - g_i^Tx)(h_j - g_j^Tx) = 0 & \forall\ (i,j) \in C\ , \quad (3)
\end{array}
\tag{P}
$$

where $x \in \mathbb{R}^n$. The index set for inequality constraints is given by $I = \{1, \dots, m_I\}$, and the index set for equality constraints is given by $E = \{1, \dots, m_E\}$. For convenience, we will sometimes equivalently write constraints (P.1) and (P.2) as $Ax = b$ and $Gx \le h$, where a_i and g_j correspond to row i of A and row j of G, respectively, reshaped as column vectors. Let $C \subseteq \{(i,j) \in I \times I \mid i < j\}$ be the set of pairs of indices (i, j) such that we have complementarity between the i^{th} and j^{th} inequality constraints.

This chapter proceeds as follows. We first construct and assess a basic relaxation of (P), then present and evaluate a number of potential improvements. After discussing the computational implications of the different relaxation models, we recommend the use of an iterative method and discuss issues related to implementation and use in practice.

3.1 The base relaxation

3.1.1 Construction

We construct an SDP relaxation of (P) according to the procedure described in section 1.1.4, beginning with lifting the problem to the variable space of $X = \left[\begin{smallmatrix}1\\x\end{smallmatrix}\right]\left[\begin{smallmatrix}1\\x\end{smallmatrix}\right]^T = \left[\begin{smallmatrix}1 & x^T\\ x & xx^T\end{smallmatrix}\right]$. Lemma 1.1.2 is employed to express constraints using the dot notation introduced in section 1.1.4.

By lemma 1.1.2, the quadratic expression $x^TQx + p^Tx + r$ can be lifted to the space of X as follows:

$$
x^TQx + p^Tx + r = \begin{bmatrix}1 & x^T\end{bmatrix}\begin{bmatrix} r & \frac{p^T}{2} \\ \frac{p}{2} & Q\end{bmatrix}\begin{bmatrix}1\\x\end{bmatrix} = \begin{bmatrix} r & \frac{p^T}{2} \\ \frac{p}{2} & Q\end{bmatrix} \cdot X\ . \tag{3.1}
$$

Similarly, the constraints can be rewritten as follows:

$$a_i^T x = b_i \qquad \Rightarrow \qquad \begin{bmatrix} b_i & -a_i^T \\ 0_{n\times 1} & 0_{n\times n} \end{bmatrix} \cdot X = 0 \ , \tag{3.2}$$

$$g_i^T x \leq h_i \qquad \Rightarrow \qquad \begin{bmatrix} h_i & -g_i^T \\ 0_{n\times 1} & 0_{n\times n} \end{bmatrix} \cdot X \geq 0 \ , \tag{3.3}$$

$$\begin{aligned} (h_i - g_i^T x)(h_j - g_j^T x) = 0 \qquad &\Rightarrow \qquad \begin{bmatrix} 1 & x \end{bmatrix} \begin{bmatrix} h_i \\ -g_i \end{bmatrix} \begin{bmatrix} h_j & -g_j^T \end{bmatrix} \begin{bmatrix} 1 \\ x \end{bmatrix} = 0 \\ &\Rightarrow \qquad \left(\begin{bmatrix} h_i \\ -g_i^T \end{bmatrix} \begin{bmatrix} h_j & -g_j^T \end{bmatrix} \right) \cdot X = 0 \ . \end{aligned} \tag{3.4}$$

We can now assemble a preliminary SDP relaxation for (P) by relaxing the rank one assumption and instead imposing the constraint $X = \left[\begin{smallmatrix} 1 & x^T \\ x & \bar{X} \end{smallmatrix}\right] \in \mathbb{S}_+^{n+1}$:

$$\begin{aligned} \min_X \quad & \begin{bmatrix} r & \frac{p^T}{2} \\ \frac{p}{2} & Q \end{bmatrix} \cdot X \\ \text{s.t.} \quad & \begin{bmatrix} b_i & -a_i^T \\ 0_{n\times 1} & 0_{n\times n} \end{bmatrix} \cdot X = 0 && \forall\ i \in E && (1) \\ & \begin{bmatrix} h_i & -g_i^T \\ 0_{n\times 1} & 0_{n\times n} \end{bmatrix} \cdot X \geq 0 && \forall\ i \in I && (2) \\ & \left(\begin{bmatrix} h_i h_j & -h_i g_j^T \\ -h_j g_i^T & g_i g_j^T \end{bmatrix} \right) \cdot X = 0 && \forall\ (i,j) \in C && (3) \\ & X \geq 0, \quad X_{00} = 1 \ . && && (4) \end{aligned} \tag{S_{base}}$$

In implementation, it is convention to symmetrize the coefficient matrices of symmetric variable matrix X, i.e. a nonsymmetric coefficient matrix M is replaced by $\frac{1}{2}(M + M^T)$. We will present models with asymmetric coefficient matrices where this serves clarity and understanding, but it should be assumed that all coefficient matrices are symmetrized before the model is passed to a solver.

3.1.2 Theory and Observations

To discuss the relation between (P) and (S_{base}), we will introduce an additional mathematical program, (R), which we will define to be the same problem as (S_{base}) but with the rank one assumption retained. That is, (R) has the form

$$
\begin{aligned}
\min_{X} \quad & \begin{bmatrix} r & \frac{p^T}{2} \\ \frac{p}{2} & Q \end{bmatrix} \cdot X \\
\text{s.t.} \quad & \begin{bmatrix} b_i & -a_i^T \\ 0_{n\times 1} & 0_{n\times n} \end{bmatrix} \cdot X = 0 && \forall\ i \in E && (1) \\
& \begin{bmatrix} h_i & -g_i^T \\ 0_{n\times 1} & 0_{n\times n} \end{bmatrix} \cdot X \geq 0 && \forall\ i \in I && (2) \\
& \left(\begin{bmatrix} h_i h_j & -h_i g_j^T \\ -h_j g_i^T & g_i g_j^T \end{bmatrix} \right) \cdot X = 0 && \forall\ (i,j) \in C && (3) \\
& rank(X) = 1, \;\; X_{00} = 1 \;\; . && && (4)
\end{aligned}
\tag{R}
$$

Lemma 3.1.1. *The dot product of two real positive semidefinite matrices is nonnegative.*

Proof. Let A and B be two positive semidefinite matrices in $\mathbb{R}^{n\times n}$. For this proof we will use two well known properties of positive semidefinite matrices. First, for a real positive semidefinite matrix B, there exist vectors $b_1, \ldots, b_{rank(B)} \in \mathbb{R}^n$ such that B has the rank decomposition

$$B = \sum_{i=1}^{rank(B)} b_i b_i^T \;\; . \tag{3.5}$$

Secondly, for positive semidefinite matrix A, $x^T A x \geq 0 \quad \forall\ x \in \mathbb{R}^n$ by definition. Using these properties, the lemma is proven as follows:

$$A \cdot B \;\; = \;\; \text{tr}(B^T A) \;\; = \;\; \text{tr}\left(\sum_{i=1}^{rank(B)} b_i b_i^T A \right) \;\; = \;\; \sum_{i=1}^{rank(B)} b_i^T A b_i \;\; \geq \;\; 0 \;\; . \tag{3.6}$$

□

Theorem 3.1.2. *Problems (P) and (R) are equivalent. There exists a one-to-one mapping between their feasible regions, and corresponding solutions achieve equivalent objective values.*
A solution $x \in feas(P)$ can be mapped to a solution $X \in feas(R)$ by the lifting $X = \left[\begin{smallmatrix} 1 & x^T \\ x & xx^T \end{smallmatrix} \right]$. A solution $\hat{X} \in feas(R)$ can be mapped to a solution $x \in feas(P)$ by the extraction $x = [X]_{2..n+1,1}$, is feasible for (P). Equations (3.7) visualize this extraction:

$$\hat{X}^* = \begin{pmatrix} X_{00} & X_{01} & \cdots & X_{0n} \\ \boxed{\begin{matrix} X_{10} \\ \vdots \\ X_{n0} \end{matrix}} & \ddots & & \vdots \\ & & \ddots & \vdots \\ & \cdots & \cdots & X_{nn} \end{pmatrix} , \qquad \hat{x}^* = \boxed{\begin{matrix} X_{10} \\ \vdots \\ X_{n0} \end{matrix}} \;\; . \tag{3.7}$$

Proof. Given x which is feasible for (P), we must show that $X = \begin{bmatrix} 1 & x^T \\ x & xx^T \end{bmatrix}$ will be feasible for (R). Note that by construction X will satisfy the constraints $rank(X) = 1, X_{00} = 1$ of (R). Consider the complementarity constraint between $(i, j) \in C$ for problem (P):

$$(h_i - g_i^T x)(h_j - g_j^T x) = 0 \quad , \tag{3.8}$$

and its lifted counterpart:

$$\left(\begin{bmatrix} h_i h_j & -h_i g_j^T \\ -h_j g_i^T & g_i g_j^T \end{bmatrix} \right) \cdot X = 0 \quad . \tag{3.9}$$

We see that for any x such that (3.8) holds, $X = \begin{bmatrix} 1 & x^T \\ x & xx^T \end{bmatrix}$ will satisfy (3.9):

$$\begin{aligned} \left(\begin{bmatrix} h_i h_j & -h_i g_j^T \\ -h_j g_i^T & g_i g_j^T \end{bmatrix} \right) \cdot X &= \left(\begin{bmatrix} h_i \\ -g_i \end{bmatrix} \begin{bmatrix} h_j & -g_j^T \end{bmatrix} \right) \cdot \left(\begin{bmatrix} 1 \\ x \end{bmatrix} \begin{bmatrix} 1 & x^T \end{bmatrix} \right) \\ &= \begin{bmatrix} 1 & x^T \end{bmatrix} \begin{bmatrix} h_i \\ -g_i \end{bmatrix} \begin{bmatrix} h_j & -g_j^T \end{bmatrix} \begin{bmatrix} 1 \\ x \end{bmatrix} \\ &= (h_i - g_i^T x)(h_j - g_j^T x) \\ &= 0 \quad . \end{aligned} \tag{3.10}$$

On the other hand, given X which is feasible for (R), we must show that extracted vector $x = [X]_{2..n+1,1}$ will be feasible for (P).

Since X is a feasible solution for (R), it is known that $X = \begin{bmatrix} 1 & x^T \\ x & xx^T \end{bmatrix} = \begin{bmatrix} 1 \\ x \end{bmatrix} \begin{bmatrix} 1 & x^T \end{bmatrix}$.

$$\begin{aligned} (h_i - g_i^T x)(h_j - g_j^T x) &= \begin{bmatrix} 1 & x^T \end{bmatrix} \begin{bmatrix} h_i \\ -g_i \end{bmatrix} \begin{bmatrix} h_j & -g_j^T \end{bmatrix} \begin{bmatrix} 1 \\ x \end{bmatrix} \\ &= \left(\begin{bmatrix} h_i h_j & -h_i g_j^T \\ -h_j g_i^T & g_i g_j^T \end{bmatrix} \right) \cdot X \\ &= 0 \quad . \end{aligned} \tag{3.11}$$

Once again, the same can be shown for the other pairs of corresponding constraints, effectively proving that there is a one-to-one mapping between the feasible region of (P) and the feasible region of the constraints which are liftings of the constraints of (R), ie.

$$
\begin{aligned}
x \in \Big\{\ x \ \Big|\ & a_i^T x = b_i && \forall\ i \in E\ , \\
& g_i^T x \le h_i && \forall\ i \in I\ , \\
& (h_i - g_i^T x)(h_j - g_j^T x) = 0 && \forall\ (i,j) \in C \quad \Big\} \\
& \Updownarrow \\
\begin{bmatrix} 1 & x^T \\ x & xx^T \end{bmatrix} \in \Big\{\ X \ \Big|\ & \begin{bmatrix} b_i & -a_i^T \\ 0_{n\times 1} & 0_{n\times n} \end{bmatrix} \cdot X = 0 && \forall\ i \in E\ , \\
& \begin{bmatrix} h_i & -g_i^T \\ 0_{n\times 1} & 0_{n\times n} \end{bmatrix} \cdot X \ge 0 && \forall\ i \in I\ , \\
& \left(\begin{bmatrix} h_i h_j & -h_i g_j^T \\ -h_j g_i^T & g_i g_j^T \end{bmatrix} \right) \cdot X = 0 && \forall\ (i,j) \in C \quad \Big\}\ .
\end{aligned}
\tag{3.12}
$$

Finally, all that remains to be shown is that the objective value obtained by x in (P) is the same as that obtained by lifted matrix X in (R). This is easily shown to be

$$
\begin{aligned}
z_P(x) &= x^T Q x + p^T x + r \\
&= Q \cdot (xx^T) + p^T x + r \\
&= \begin{bmatrix} r & \frac{1}{2}p^T \\ \frac{1}{2}p & Q \end{bmatrix} \cdot \begin{bmatrix} 1 & x^T \\ x & xx^T \end{bmatrix} \\
&= z_R(X)\ ,
\end{aligned}
\tag{3.13}
$$

and so a one-to-one mapping exists with corresponding solutions achieving the same values.

□

Corollary 3.1.3. *Given X which is feasible for (S_{base}) and is rank one, there exists a mapping to a feasible solution of (P) with equivalent value.*

Proof. Letting $feas(P)$ denote the set of feasible solutions to a problem (P), we observe that $feas(R) = feas(S) \cap \{\hat{X} \text{ s.t. } rank(\hat{X}) = 1\}$. Then we have $X \in feas(R)$, and as per Theorem 3.1.2 we can map it to x which is feasible for (P) and achieves the same objective value. □

Corollary 3.1.4. *If (P) is feasible, (S_{base}) will be feasible.*

Proof. Let x_{feas} be a feasible solution to (P). Then we can map it to a solution (S_{base}) as shown in Theorem (3.1.2). □

Example 3.1.5. While (S_{base}) is a valid SDP relaxation for any (P), we do not necessarily expect any particular degree of tightness. In particular, for the case of a nonconvex quadratic objective

function, the model (S_{base}) can easily be unbounded. This is due to the fact that the block of matrix X which serves as a proxy for the quadratic terms xx^T is relatively unconstrained.

For example, let us consider the following problem:

$$\begin{aligned} \max_{x,y,z} \quad & yz \\ \text{s.t.} \quad & 0 \leq x, y, z \leq 1 \qquad (1) \\ & xy = 0 \qquad (2) \\ & xz = 0 \ . \qquad (3) \end{aligned} \tag{P^2_{Ex}}$$

We can solve this problem analytically by first observing that y and z can only be nonzero if we fix $x = 0$. Then the problem reduces to

$$\begin{aligned} \max_{y,z} \quad & yz \\ \text{s.t.} \quad & 0 \leq y, z \leq 1 \ , \qquad (1) \end{aligned} \tag{$P^2_{Ex}{}'$}$$

with optimal value 1 achieved by solution $(x, y, z) = (0, 1, 1)$.

However, let us consider the SDP relaxation given by relaxing this problem as in $(P) \rightarrow (S_{base})$:

$$\begin{aligned} \max_{X} \quad & X_{23} \\ \text{s.t.} \quad & 0 \leq X_{01}, X_{02}, X_{03} \leq 1 \qquad (1) \\ & X_{12} = 0 \qquad (2) \\ & X_{13} = 0 \qquad (3) \\ & X = \begin{bmatrix} 1 & X_{01} & X_{02} & X_{03} \\ X_{01} & X_{11} & X_{12} & X_{13} \\ X_{02} & X_{12} & X_{22} & X_{23} \\ X_{03} & X_{13} & X_{23} & X_{33} \end{bmatrix} \succeq 0 \ . \qquad (4) \end{aligned} \tag{S_{ex1}}$$

For any $v \geq 1$, the matrix

$$X = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & v & v \\ 1 & 0 & v & v \end{bmatrix} \tag{3.14}$$

is a feasible solution achieving objective value v. Since v can be increased arbitrarily, the relaxation is unbounded.

This example illustrates an important point. In the problem (P^2_{Ex}), the objective function yz is bounded implicitly by the constraints $0 \leq y \leq 1,\ 0 \leq z \leq 1$. On the other hand, in the SDP relaxation (S_{ex1}) the corresponding variable X_{23} is constrained only as part of the structural constraint $X \succeq 0$, and there is enough flexibility overall that a feasible X matrix can be constructed to accommodate any desired value of X_{23}.

Theorem 3.1.6. *For any real positive semidefinite matrix* $\begin{bmatrix} r & \frac{p^T}{2} \\ \frac{p}{2} & Q \end{bmatrix}$, *the problem*

$$\begin{aligned} \min_X \quad & \begin{bmatrix} r & \frac{p^T}{2} \\ \frac{p}{2} & Q \end{bmatrix} \cdot \begin{bmatrix} 1 & x^T \\ x & \bar{X} \end{bmatrix} \\ s.t. \quad & \begin{bmatrix} 1 & x^T \\ x & \bar{X} \end{bmatrix} \succeq 0 \end{aligned} \tag{3.15}$$

is feasible and bounded.

Proof. Any $x \in \mathbb{R}^n$ and $\bar{X} = xx^T$ will yield a feasible solution. Since $\begin{bmatrix} 1 & x^T \\ x & \bar{X} \end{bmatrix} \in \mathbb{S}^{n+1}_+$ for any feasible solution, Lemma 3.1.1 gives a lower bound of 0 on the value of the objective function on the feasible set. □

The condition stated for Theorem 3.1.6 is more restrictive than we would like. For example, it is possible for $\left[\begin{smallmatrix} r & \frac{p^T}{2} \\ \frac{p}{2} & Q \end{smallmatrix}\right]$ to be indefinite even if $x^TQx + p^Tx + r$ is a strictly convex expression. We would prefer to draw a connection between the eigenvalues of Q and the boundedness of the semidefinite relaxation. The following lemma does this, albeit while requiring a stricter condition on Q, positive definiteness.

Lemma 3.1.7. *Given a positive definite matrix* $Q \in \mathbb{S}^n_{++}$, $p \in \mathbb{R}^n$, $r \in \mathbb{R}$, *there exists a finite* $c \in \mathbb{R}$ *such that*

$$M(c) = \begin{bmatrix} r + c & \frac{p^T}{2} \\ \frac{p}{2} & Q \end{bmatrix} \succ 0 \quad . \tag{3.16}$$

Proof. Given $M(c) \in \mathbb{R}^{n+1 \times n+1}$ and index sets $I, J \subseteq \{1, \ldots, n+1\}$, $det_{I,J}(M(c))$ denotes the determinant of the submatrix given by selecting rows I and columns J of $M(c)$. The principal minors of $M(c)$ are given by $det_{I,I}(M(c))$ for some $I \subseteq \{1, \ldots, n\}$, and the leading principal minors are those principal minors for which $I = \{1, \ldots, k\}$ for some $k \leq n+1$. To make $M(c)$ positive definite, it is sufficient to identify a $c \in \mathbb{R}$ such that all the leading principal minors of $M(c)$ are positive, i.e.

$$\det_{I,I}(M(c)) > 0 \quad \forall\ I = \{1, \ldots, k\},\ \ k \leq n+1 \quad . \tag{3.17}$$

Every leading principal minor will have $1 \in I$. First, consider the special case where $I = \{1\}$.

$$\det_{\{1\},\{1\}}(M(c)) = r + c > 0 \qquad \Leftrightarrow \qquad c > -r \ . \tag{3.18}$$

In the remaining cases, where $1 \in I$ and $|I| \geq 2$, we can write

$$\begin{aligned} \det_{I,I}(M(c)) &= \sum_{k=1}^{|I|} (-1)^{k+1} X_{1,I_k} \det_{I\setminus\{1\},I\setminus\{I_k\}}(M(c)) \\ &= (r+c) \det_{I\setminus\{1\},I\setminus\{1\}}(M(c)) + \sum_{k=2}^{|I|} (-1)^{k+1} X_{1,I_k} \det_{I\setminus\{1\},I\setminus\{I_k\}}(M(c)) \ . \end{aligned} \tag{3.19}$$

The principal minor $\det_{I\setminus\{1\},I\setminus\{1\}}(M(c))$ is equivalent to one of the principal minors of Q and is therefore positive, so the necessary condition $det_{I,I}(M(c)) > 0$ can be expressed as

$$c > \frac{\sum_{k=2}^{|I|} (-1)^{k} X_{1,I_k} \det_{I\setminus\{1\},I\setminus\{I_k\}}(M(c))}{\det_{I\setminus\{1\},I\setminus\{1\}}(M(c))} - r \ . \tag{3.20}$$

Furthermore, the right hand side of this inequality does not in fact depend on c since all the determinants involved omit the first row of $M(c)$.

Therefore, $M(c)$ will be positive definite for any c such that

$$c > -r \ , \qquad c > -r + \max_{\substack{I\subseteq\{1\ldots n+1\} \\ 1\in I,\ |I|\geq 2}} \frac{\sum_{k=2}^{|I|} (-1)^{k} X_{1,I_k} \det_{I\setminus\{1\},I\setminus\{I_k\}}(M(c))}{\det_{I\setminus\{1\},I\setminus\{1\}}(M(c))} \ . \tag{3.21}$$

Since the maximum is being taken over a finite number of sets $|I|$, and the bound on c will be finite in each case, there exists a finite and calculable c such that $M(c) \succ 0$.

□

Theorem 3.1.8. *For any real positive definite matrix Q, the problem*

$$\begin{aligned} \min_{X} \quad & \begin{bmatrix} r & \frac{p^T}{2} \\ \frac{p}{2} & Q \end{bmatrix} \cdot \begin{bmatrix} 1 & x^T \\ x & \bar{X} \end{bmatrix} \\ s.t. \quad & \begin{bmatrix} 1 & x^T \\ x & \bar{X} \end{bmatrix} \succeq 0 \end{aligned} \tag{3.22}$$

is feasible and bounded.

Proof. Feasibility is as in Theorem 3.1.6. To show boundedness, we use the fact that by Lemma 3.1.7, there exists a finite real c such that $\begin{bmatrix} r+c & \frac{p^T}{2} \\ \frac{p}{2} & Q \end{bmatrix} \succ 0$. Then, for such a c, we can rewrite the objective as follows and apply Lemma 3.1.1 to obtain

$$\begin{bmatrix} r & \frac{p^T}{2} \\ \frac{p}{2} & Q \end{bmatrix} \cdot \begin{bmatrix} 1 & x^T \\ x & \bar{X} \end{bmatrix} = \begin{bmatrix} r+c & \frac{p^T}{2} \\ \frac{p}{2} & Q \end{bmatrix} \cdot \begin{bmatrix} 1 & x^T \\ x & \bar{X} \end{bmatrix} - c \geq -c \quad . \tag{3.23}$$

□

Theorem 3.1.9. *Consider the following problem:*

$$\begin{aligned} \min_X \quad & \begin{bmatrix} r & \frac{p^T}{2} \\ \frac{p}{2} & Q \end{bmatrix} \cdot \begin{bmatrix} 1 & x^T \\ x & \bar{X} \end{bmatrix} \\ s.t. \quad & A_i \cdot X = b_i \qquad\qquad i \in E \\ & G_i \cdot X \leq h_i \qquad\qquad i \in I \\ & X = \begin{bmatrix} 1 & x^T \\ x & \bar{X} \end{bmatrix} \succeq 0 \quad . \end{aligned} \tag{3.24}$$

Assuming the problem is feasible, and without making any assumptions about the positive semidefiniteness of Q, if the constraints of the feasible region ensure that diagonal elements of $\bar{X}$ are bounded above by some finite U, i.e. $\bar{X}_{ii} \leq U$ for every feasible solution $x, \bar{X}$, then the problem will be bounded.

Proof. It is known that real diagonally dominant matrices with non-negative diagonal entries will be positive semidefinite. For arbitrary $M = \begin{bmatrix} r & \frac{p^T}{2} \\ \frac{p}{2} & Q \end{bmatrix}$, there exists a nonnegative scalar ϵ such that $M + \epsilon I$ is diagonally dominant with positive diagonal entries, and therefore positive semidefinite. Then we can rewrite the objective as follows and apply Lemma 3.1.1 to obtain a lower bound over the feasible region of the problem:

$$\begin{aligned} \begin{bmatrix} r & \frac{p^T}{2} \\ \frac{p}{2} & Q \end{bmatrix} \cdot \begin{bmatrix} 1 & x^T \\ x & \bar{X} \end{bmatrix} &= \begin{bmatrix} r+\epsilon & \frac{p^T}{2} \\ \frac{p}{2} & Q+\epsilon I \end{bmatrix} \cdot \begin{bmatrix} 1 & x^T \\ x & \bar{X} \end{bmatrix} - \epsilon(1 + \sum_{i=1}^{n} X_{ii}) \\ &\geq -\epsilon(1 + \sum_{i=1}^{n} X_{ii}) \\ &\geq -\epsilon(1 + nU) \quad . \end{aligned} \tag{3.25}$$

□

3.1.3 Test problems: MacMPEC

(TODO: Explain/show why no MacMPEC - limitations, motivate creation/use of PyQPECgen)

3.1.4 Test problems: QPECgen

Before presenting our first results, we must introduce the set of problems which we will use throughout this thesis to evaluate the effectiveness of SDP relaxations and related methods. The problems used are generated using PyQPECgen, our Python implementation of test problem generator QPECgen, which will be introduced in more detail in section 5.2. We generate problems of the BOX-QPEC type with convex quadratic objective functions, and problems of type FULL-BOX-QPEC with nonconvex quadratic objective functions. As discussed in section 5.2, FULL-BOX-QPEC is the only problem type generated by PyQPECgen which guarantees boundedness of the problem for nonconvex quadratic objective functions.

Each problem is generated as a QPLCC having approximately 20, 50, or 100 variables. We generate 12 problems in each size category $N \in \{20, 50, 100\}$ 6 from BOX-QPEC, and 6 from FULL-BOX-QPEC. Problems are generated according to the parameters of Table 3.1, along with the default parameters shown in 5.1. Names are assigned to problems according to the formula {B/F}{N}n{i}, identifying the type (B for BOX-QPEC, F for FULL-BOX-QPEC), size category N, and generation number i.

Table 3.1 Parameters altered from default to generate problems of each type, where $N \in \{20, 50, 100\}$ is the desired problem size category

	Case	
Parameter	BOX-QPEC	FULL-BOX-QPEC
qpec_type	200	201
n	$0.5N$	$0.4N$
m	$0.25N$	$0.2N$
l	$0.25N$	$0.2N$
convex_f	True	False
second_deg	An integer randomly selected from [0, m]	
# problems generated per N	6	6

The global MINLP solver BARON is used in advance to determine each problem's global optimal value so that metrics such as optimality gaps can be computed for our methods. Tables 3.2-3.4 show the results of solving all test problems to global optimality using BARON. Both MIP and NLP formulations were tested and the faster solve time (generally for the MIP formulation) was

recorded as t_B. Times will be used to give perspective to the time taken by our own methods.

The symbol † marks those problems for which BARON could not prove global optimality within 6 hours. In these cases it is quite possible that BARON has found the global optimum but simply can't prove it, but even if this is not the case, SDP gaps calculated against the best feasible solution will serve as upper bounds against the gap to the true global optimal solution.

Table 3.2 Global optimal results found using BARON for problems with about 20 variables.

	Problem	$(n, \|C\|)$	z^*	t_B (s)
convex objectives	B20n0	(24, 9)	-167.46	0.08
	B20n1	(21, 6)	-156.11	0.05
	B20n2	(24, 9)	-1202.5	0.19
	B20n3	(21, 6)	-229.23	0.07
	B20n4	(21, 6)	-614.43	0.09
	B20n5	(22, 7)	-265.04	0.08

	Problem	$(n, \|C\|)$	z^*	t_B (s)
nonconvex objectives	F20n0	(20, 8)	-487.56	0.07
	F20n1	(20, 8)	-132.34	0.21
	F20n2	(20, 8)	-233.10	0.1
	F20n3	(20, 8)	-96.345	0.12
	F20n4	(20, 8)	-38.382	0.3
	F20n5	(20, 8)	-235.07	0.23

Table 3.3 Global optimal results found using BARON for problems with about 50 variables.

	Problem	$(n, \|C\|)$	z^*	t_B (s)
convex objectives	B50n0	(56, 19)	-2568.1	1.9
	B50n1	(59, 22)	-1651.6	7.61
	B50n2	(59, 22)	-1769.7	8.09
	B50n3	(56, 19)	-3158.4	2.48
	B50n4	(56, 19)	-730.80	3.79
	B50n5	(54, 17)	-662.69	2.27

	Problem	$(n, \|C\|)$	z^*	t_B (s)
nonconvex objectives	F50n0	(50, 20)	-414.23	33.28
	F50n1	(50, 20)	-624.92	66.30
	F50n2	(50, 20)	-467.70	42.64
	F50n3	(50, 20)	-2649.6	14.38
	F50n4	(50, 20)	-552.23	2.99
	F50n5	(50, 20)	-580.06	36.14

Table 3.4 Global optimal results found using BARON for problems with about 100 variables.

	Problem	$(n, \|C\|)$	z^*	t_B (s)	
convex objectives	B100n0	(104, 29)	-2531.8	6618.1	
	B100n1	(108, 33)	-2701.3	6460.5	
	B100n2	(107, 32)	-2233.8	1828.5	
	B100n3	(117, 42)	-2978.1	2043.6	
	B100n4	(115, 40)	-5361.1	1235.7	
	B100n5	(117, 42)	-2008.0	1475.1	

	Problem	$(n, \|C\|)$	z^*	t_B (s)	
nonconvex objectives	F100n0	(100, 40)	-4921.8	21600.	†
	F100n1	(100, 40)	-2617.2	21600.	†
	F100n2	(100, 40)	-894.66	6411.	
	F100n3	(100, 40)	-4237.7	21600.	†
	F100n4	(100, 40)	-3118.3	10384.	
	F100n5	(100, 40)	-1098.7	21600.	†

We observe that many problems in the 20 variable category are solved in fractions of a second, while only 5 of 12 problems in the 100 variable category could be solved to global optimality by BARON in less than 6 hours, demonstrating the escalation in difficulty as QPLCC problems become large. We also note that problems with nonconvex objective functions in general take longer to solve to global optimality, which is as expected.

3.1.5 Evaluation of (S_{base})

Throughout this chapter and the next, we will discuss optimality gaps for SDP relaxations of QPLCCs.

We will define the following function for measuring relative gap:

$$gap_S(z) = \frac{z^* - z}{|z^*|} \quad , \tag{3.26}$$

where z^* is the QPLCC's optimal value as found by BARON. The metric gap_S is appropriate to evaluate the quality of an SDP bound since an SDP bound z_S will be an underestimator of z^*.

Table 3.5 shows the effectiveness of (S_{base}) for the test problem set. The SDP relaxations are solved on a dual processor Intel(R) Xeon(R) X5675 @ 3.07GHz with 96 Gb of RAM using Matlab solver SDPT3 (Tütüncü et al. (2001); Toh et al. (2006)) using our Python-to-SDPT3 interface PySDPT3glue which is detailed in section 5.3.

Only the time used by the solver is recorded in column t_S because some software components such as the relaxation step and the PySDPT3glue interface are designed to maximize flexibility, clarity, and ease of experimental design rather than efficiency. The single largest use of time is that required to export the problem from Python as a Matlab .mat file, an unavoidable step when passing a problem between the two languages, but an interested party willing to cement in a particular method could write a pure Matlab implementation with very efficient solve preparation.

As in Tables 3.3-3.4, † denotes those problems for which the true global optimum was not proven by BARON and the best known solution is used instead. For the problems considered, each problem's best known solution $\hat{z}_B$ is negative and thus bounded by $z^* < \hat{z}_B \leq 0$. We can then treat these cases by calculating gap_S against $\hat{z}_B$ rather than z^*, because this will give an upper bound on the gap (i.e. a worst case gap) to the unknown true optimum.

Table 3.5 Optimality gaps and times for the S_{base} SDP relaxation formulation.

		S_{base}			S_{base}			S_{base}			
	Problem	gap_S %	t_S (s)	Problem	gap_S %	t_S (s)	Problem	gap_S %	t_S (s)	
convex objectives	B20n0	2.66%	1.33	B50n0	0.08%	1.73	B100n0	0.52%	3.24	
	B20n1	0.01%	1.26	B50n1	2.75%	1.93	B100n1	0.39%	3.42	
	B20n2	0.40%	1.35	B50n2	4.78%	1.89	B100n2	0.07%	3.98	
	B20n3	2.83%	1.74	B50n3	0.15%	2.48	B100n3	0.24%	4.62	
	B20n4	0.09%	1.26	B50n4	1.22%	3.79	B100n4	0.13%	4.31	
	B20n5	38.24%	1.11	B50n5	2.04%	2.27	B100n5	0.67%	4.13	
nonconvex objectives	F20n0	0.00%	1.34	F50n0	44.70%	1.59	F100n0	0.36 %	2.15	†
	F20n1	0.42%	1.32	F50n1	6.72%	1.54	F100n1	1.09%	2.24	†
	F20n2	0.00%	1.21	F50n2	4.51%	1.31	F100n2	2.11%	2.12	
	F20n3	0.22%	1.24	F50n3	0.02%	1.51	F100n3	0.20%	2.3	†
	F20n4	1.07%	1.18	F50n4	0.00%	1.22	F100n4	0.09%	2.5	
	F20n5	0.40%	1.33	F50n5	0.39%	1.33	F100n5	1.55%	2.18	†

3.2 Tightening constraints and modeling alternatives

While (S_{base}) performed well for many problems, there is room for improvement in many cases. In this section we consider what additional constraints might be used to improve the tightness of the SDP relaxation. As we observed in Theorem 3.1.9, bounding the diagonal of the X matrix is one way to ensure boundedness of the relaxation, and tighter bounds on the diagonal elements may lead to tighter relaxations. More generally, we hope to devise valid constraints which are quadratic in the variables of the original problem, so that the feasible region of the used SDP relaxation will be a more effective proxy for that of (P).

For example, in the case of Example 3.1.5, the ideal feasible region for a semidefinite relaxation of (P^2_{Ex}) would be

$$\mathcal{F}_{SDP} = \left\{ \begin{bmatrix} 1 & x & 0 & 0 \\ x & x^2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \forall\, x \in [0,1] \right\} \cup \left\{ \begin{bmatrix} 1 & 0 & y & z \\ 0 & 0 & 0 & 0 \\ y & 0 & y^2 & yz \\ z & 0 & yz & z^2 \end{bmatrix}, \forall\, y, z \in [0,1] \right\} , \tag{3.27}$$

To improve relaxation (S_{ex1}), we need to identify valid cuts which will bring the feasible region of (S_{ex1}) closer to the convex hull of $\mathcal{F}$, denoted $conv(\mathcal{F})$. Note that the feasible region cannot be

made any tighter than $conv(\mathcal{F})$ because the feasible region of a semidefinite program is convex by construction, having a semidefinite cone constraint and otherwise linear constraints.

In this case, (S_{ex1}) can be made an exact SDP relaxation by the addition of the constraint

$$(1-y)z \geq 0 \qquad \Rightarrow \qquad \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot X \geq 0 \ , \tag{3.28}$$

which is a valid cut derived from constraints $y \leq 1$ and $z \geq 0$ of (P^2_{Ex}). This is an example of a Sherali-Adams constraint, the first type of tightening constraint we will explain.

3.2.1 Sherali-Adams constraints

Sherali-Adams constraints are rooted in the notion that the product of two non-negative quantities will be non-negative. For any $i \in I$, $j \in I$, the (i, j) Sherali-Adams constraint is given by

$$(h_i - g_i^T x)(h_j - g_j^T x) \geq 0 \ . \tag{3.29}$$

Similarly to (3.4), this constraint can be modeled as

$$\left(\begin{bmatrix} h_i \\ -g_i \end{bmatrix} \begin{bmatrix} h_j & -g_j^T \end{bmatrix} \right) \cdot X \geq 0 \ . \tag{3.30}$$

We impose this constraint for only those $i, j \in I$ which satisfy $i < j$ and $(i, j) \notin C$. The constraint for (i, j) is equivalent to that for (j, i), the (i, i) case is trivially satisfied (see lemma 3.2.1), and when $(i, j) \in C$ the (i, j) Sherali-Adams constraint is redundant in the corresponding complementarity constraint (3.4).

Lemma 3.2.1. *The Sherali-Adams constraint between an inequality constraint and itself will be trivially satisfied by any positive semidefinite X.*

Proof. By the definition of $X \geq 0$,

$$\left(\begin{bmatrix} h_i \\ -g_i \end{bmatrix} \begin{bmatrix} h_i & -g_i^T \end{bmatrix} \right) \cdot X = \begin{bmatrix} h_i & -g_i^T \end{bmatrix} X \begin{bmatrix} h_i \\ -g_i \end{bmatrix} \geq 0 \ . \tag{3.31}$$

□

Example 3.2.2 demonstrates the use of a Sherali-Adams constraint in a simple example.

Example 3.2.2. Consider the nonconvex QP problem and potential semidefinite programming relaxation:

$$\begin{aligned} &\max\ y = x^2 && &&\max\ y \\ &\text{s.t. } -1 \le x \le 2\ , \quad (P^3_{Ex}) && &&\text{s.t. } -1 \le x \le 2 \qquad (S^3_{Ex}) \\ & && &&X = \begin{bmatrix} 1 & x \\ x & y \end{bmatrix} \succeq 0\ . \end{aligned}$$

Figures 3.1 and 3.2 visualize the solution space of (P_0) and (S_0), and we observe that (S_0) is unbounded. However, reinforcing (S_0) by adding the Sherali-Adams constraint

$$(2-x)(x+1) \ge 0 \quad \Rightarrow \quad y \le x+2 \quad \Rightarrow \quad \begin{bmatrix} 2 & 0.5 \\ 0.5 & -1 \end{bmatrix} \cdot X \ge 0 \tag{3.32}$$

bounds the relaxation by enforcing $y \le x + 2 \le 4$. Calling this reinforced relaxation (S_{0+}), the new solution space is shown by Figure 3.3 and (S_{0+}) is an exact relaxation of (P_0) since they have the same optimal value. The feasible regions of example problem P_0 and two valid SDP relaxations, demonstrating the impact of Sherali-Adams constraints.

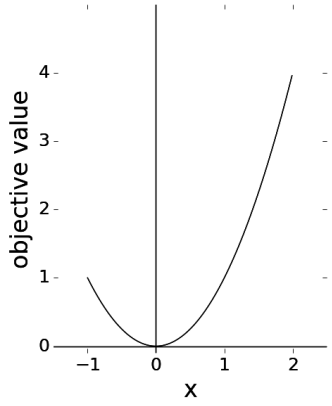

Figure 3.1 $feas(P_0)$ is a segment of parabola $y = x^2$.

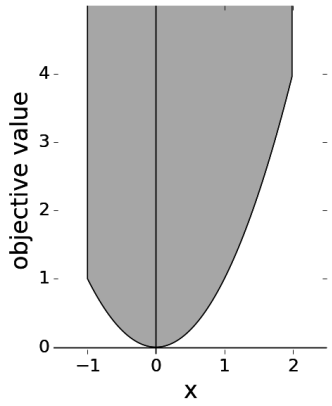

Figure 3.2 Naive relaxation (S_0) is unbounded.

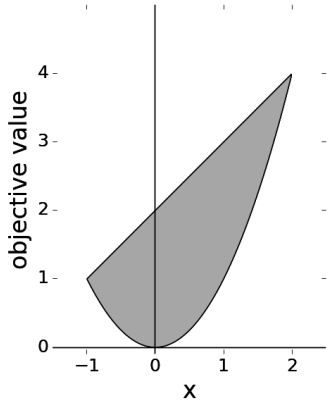

Figure 3.3 $feas(S_{0+})$ is the convex hull of $feas(P_0)$.

There exists a connection between Sherali-Adams constraints and McCormick inequalities. Given explicit variable bound constraints

$$\begin{aligned} &x_1 \ge l_1\ , \quad (a) && &&x_2 \ge l_2\ , \quad (c) \\ &x_1 \le u_1\ , \quad (b) && &&x_2 \le u_2\ , \quad (d) \end{aligned} \tag{3.33}$$

the McCormick envelope for variables x_1, x_2 and quadratic proxy term $\bar{X}_{12}$ is given by the constraints

$$\begin{aligned} \bar{X}_{12} &\geq \ell_2 x_1 + \ell_1 x_2 - \ell_1 \ell_2 \ , \qquad (a) \\ \bar{X}_{12} &\leq u_2 x_1 + \ell_1 x_2 - \ell_1 u_2 \ , \qquad (b) \\ \bar{X}_{12} &\leq \ell_2 x_1 + u_1 x_2 - u_1 \ell_2 \ , \qquad (c) \\ \bar{X}_{12} &\geq u_2 x_1 + u_1 x_2 - u_1 u_2 \ . \qquad (d) \end{aligned} \tag{3.34}$$

These constraints are equivalent to the Sherali-Adams constraints generated by inequality pairs (3.33a) and (3.33c), (3.33a) and (3.33d), (3.33b) and (3.33c), and (3.33b) and (3.33d), respectively. Therefore, if explicit variable bound constraints are present in the inequalities of (P), the constraints of the McCormick envelope will be included among the Sherali-Adams constraints generated for I.

The McCormick envelope is often applied to mixed integer nonlinear programs (MINLPs), and is a crucial element of global MINLP solvers such as BARON which use the spatial branch and bound method. An overview of convex envelopes can be found in Costa and Liberti (2012), and an in-depth discussion of convex envelopes and spatial branch and bound can be found in the book of Tawarmalani and Sahinidis (2002).

3.2.2 Enhanced equality constraints

This second constraint type is rooted in the fundamental notion that an equality remains valid when multiplied on both sides by any quantity. The name takes a cue from Ševčovič and Trnovská (2014), who proposed an 'enhanced semidefinite relaxation method' by which all constraints of this type are imposed. This constraint type also features in the SDP model used in Braun and Mitchell (2005).

For any variable x_i and equality constraint $a_j^T x = b_j$, the quadratic equality constraint

$$x_i(b_j - a_j^T x) = 0 \qquad \Rightarrow \qquad \begin{bmatrix} b_j & -a_j^T \end{bmatrix} X e_{i+1} = 0 \tag{3.35}$$

is valid where $e_j \in \mathbb{R}^{n+1}$, $[e_j]_k = \begin{cases} 1 & \text{if } k = j+1, \\ 0 & \text{otherwise.} \end{cases}$

It is trivial to see that constraint (3.35) is valid, but the impact it may have as a cut in an SDP relaxation is not immediately obvious. The motivation for its use comes from noting that $X \succeq 0$ has rank one if and only if columns $2, \ldots, n+1$ of X are scalar multiples of its first column, specifically $X_{i,:} = X_{i,0} * X_{0,:}$. In general, this relationship can't be enforced directly using tractable constraints, but tightening towards this end may be possible indirectly by thinking about how properties of column 0 should be reflected in column i. In this case, we observe that if column 0 of X is con-

strained to lie on some hyperplane defined by $\left\{z \in \mathbb{R}^{n+1} \text{ s.t. } \begin{bmatrix} h_j & -g_j^T \end{bmatrix} z = 0\right\}$, then we can impose that column i must as well.

In practice, diminishing returns are common when imposing this constraint type for a given j and many i. Furthermore, the constraints are not necessarily independent of one another and numerical difficulties often arise in practice when solving the SDP relaxation with many of these constraints imposed redundantly.

3.2.3 Aggregated equality constraints

Either as an alternative to or in combination with constraints of the type (3.35), we can also consider the valid constraint

$$x^T A^T (b - Ax) = 0 \ , \tag{3.36}$$

written in terms of X as

$$\begin{bmatrix} 0 & 0 \\ A^T b & -A^T A \end{bmatrix} \cdot X = 0 \ . \tag{3.37}$$

This can be seen as the weighted aggregation of enhanced equality constraints:

$$x^T A^T (Ax - b) = 0 \quad \Rightarrow \quad \sum_{i=1}^{n} \sum_{j=1}^{|I|} A_{ji} x_i (b_j - a_j^T x) = 0 \quad \Rightarrow \quad \sum_{i=1}^{n} \sum_{j=1}^{|E|} A_{ji} \begin{bmatrix} b_j & -a_j^T \end{bmatrix} X e_{i+1} = 0 \ . \tag{3.38}$$

Having made this observation, another alternative may be to consider the case where all enhanced equality constraints are aggregated with equal weight:

$$\sum_{i=1}^{n} \sum_{j=1}^{|E|} \begin{bmatrix} b_j & -a_j^T \end{bmatrix} X e_{i+1} = 0 \ . \tag{3.39}$$

Constraints (3.38) and (3.39) will both be redundant if all $n|E|$ enhanced equality constraints are individually imposed, but when we are unwilling to impose that many constraints then these aggregated constraints may give some benefit for a very cheap cost.

3.3 A 'full' model

We will now say that for (P), a 'full' SDP relaxation (with regards to the tightening measures discussed in this thesis) would be given by:

$$
\begin{aligned}
\min_{X} \quad & \begin{bmatrix} r & \frac{p^T}{2} \\ \frac{p}{2} & Q \end{bmatrix} \cdot X \\
\text{s.t.} \quad & \begin{bmatrix} b_i & -a_i^T \end{bmatrix} X e_j = 0 && \forall\ i \in E,\ j = 0, \ldots, n && (1) \\
& \begin{bmatrix} h_i & -g_i^T \end{bmatrix} X e_0 \geq 0 && \forall\ i \in I && (2) \\
& \begin{bmatrix} h_i & -g_i^T \end{bmatrix} X \begin{bmatrix} h_j \\ -g_j^T \end{bmatrix} = 0 && \forall\ (i,j) \in C && (3) \\
& \begin{bmatrix} h_i & -g_i^T \end{bmatrix} X \begin{bmatrix} h_j \\ -g_j^T \end{bmatrix} \geq 0 && \forall\ 0 \leq i < j \leq n_i,\ (i,j) \notin C && (4) \\
& X \geq 0, \quad X_{00} = 1\ . && && (5)
\end{aligned}
\qquad (S_{full})
$$

Aggregated constraints 3.38 and 3.39 are redundant in $(S_{full}.1)$ and are omitted.

We can extend some of our earlier theoretical results. Let the supplementary mathematical program (R_{full}) be defined to be the same problem as (S_{full}) but with the rank one assumption retained. In other words, with constraint $(S_{full}.5)$ replaced by the rank one structure constraint $X = \left[\begin{smallmatrix} 1 & x^T \\ x & xx^T \end{smallmatrix}\right]$. Then theoretical results 3.1.2-3.1.4 can be similarly proven if (R) and (S_{base}) are replaced by (R_{full}) and (S_{full}), because we have only added valid constraints which are redundant in the original problem and in (R_{full}). This further extends to any model which uses all the constraints of (S_{base}) supplemented with some or all of the constraints of (S_{full}).

The trouble with (S_{full})

Although (S_{full}) is theoretically valid, upon testing this model it quickly becomes apparent that (S_{full}) is an impractical relaxation for problems of even modest size. SDP relaxations of problems with as few as 20 variables return a host of non-convergence errors: maximum number of iterations reached, progress in relative gap or infeasibility is bad, primal infeasibility has deteriorated too much, etc.

We hypothesize that this is because $(S_{full}.1)$ imposes $O(n|E|)$ equality constraints, many of which are redundant in the rest of the problem. These redundant constraints are harmless theoretically, but in our experience impose constraints which have a very small amount of numerical error built into them. The effect is that they slice away at the interior of the SDP model and create difficulty

for an interior point solver.

3.4 A middle ground model and iterative framework

To construct a model which is computationally manageable, if not as theoretically strong as (S_{full}), we will consider a model (S_{heur}) and define it as

$$
\begin{aligned}
\min_{X} \quad & \begin{bmatrix} r & \frac{p^T}{2} \\ \frac{p}{2} & Q \end{bmatrix} \cdot X \\
\text{s.t.} \quad & \begin{bmatrix} b_i & -a_i^T \end{bmatrix} X e_0 = 0 \qquad \forall\, i \in E \qquad (1) \\
& \begin{bmatrix} h_i & -g_i^T \end{bmatrix} X e_0 \geq 0 \qquad \forall\, i \in I \qquad (2) \\
& \begin{bmatrix} h_i & -g_i^T \end{bmatrix} X \begin{bmatrix} h_j \\ -g_j^T \end{bmatrix} = 0 \quad \forall\, (i,j) \in C \quad (3) \\
& \begin{bmatrix} 0 & 0 \\ A^T b & -A^T A \end{bmatrix} \cdot X = 0 \qquad (4) \\
& X \succeq 0 \ . \qquad (5)
\end{aligned}
\qquad (S_{heur})
$$

This model is relaxed from (S_{full}) in two ways. First, the single aggregated equality constraint (S_{heur}.1) is used in place of the $O(n|E|)$ enhanced equality constraints imposed in (S_{full}.1), and second, the Sherali-Adams constraints seen in constraint (4) of (S_{full}) have been removed for the moment. Large numbers of inequality constraints are generally less problematic than large numbers of equality constraints, but since there are $O(|I|^2)$ such constraints we will leave them out initially to prevent the constraints from falsely slicing through parts of the ideal feasible region, (R), due to numerical error. The model (S_{heur}) contains only $O(n + |I| + |E|)$ linear constraints.

Beginning with Table 3.6, we will use CVXOPT to compute solutions to SDP problems. CVXOPT is an SQLP solver package developed for Python by Vandenberghe (2010) and Andersen et al. (2013), with much of the underlying computation performed using C. In general we find that CVXOPT has less overhead for very small problems such as the $N = 20$ group, but suffers from much worse computational complexity as the number of variables increases. Although we again use a dual processor Intel(R) Xeon(R) X5675 @ 3.07GHz with 96 Gb of RAM, it is not taken advantage of this time and our solves commonly take 100+ times longer than with SDPT3 for $N = 100$ problems. Nonetheless, we will begin using CVXOPT at this point for the sake of integrity, as we have observed that in practice it is the more stable algorithm for the (S_{heur}) model and the iterative models to come. The column t_C will denote the solve time taken by CVXOPT. Remarkably, in chapter 4 we will observe that a global solution method assisted by this SDP solve can be competitive despite

the slow solution times for these problems.

Table 3.6 Optimality gaps and times for the (S_{heur}) SDP relaxation formulation.

		S^0_{iter}			S^0_{iter}			S^0_{iter}	
	Problem	gap_S %	t_C (s)	Problem	gap_S %	t_C (s)	Problem	gap_S %	t_C (s)
convex objectives	B20n0	2.46%	0.69	B50n0	0.05%	39.23	B100n0	0.48%	860.75
	B20n1	0.00%	0.44	B50n1	2.72%	57.95	B100n1	0.28%	1075.23
	B20n2	0.20%	1.41	B50n2	4.76%	71.44	B100n2	0.00%	984.49
	B20n3	2.61%	0.45	B50n3	0.14%	44.19	B100n3	0.09%	2113.35
	B20n4	0.09%	0.42	B50n4	0.88%	73.62	B100n4	0.10%	1613.11
	B20n5	38.23%	0.64	B50n5	1.98%	39.57	B100n5	0.59%	2279.48
nonconvex objectives	F20n0	0.00%	0.48	F50n0	44.47%	34.23	F100n0	0.13%	745.77
	F20n1	0.37%	0.39	F50n1	6.58%	19.97	F100n1	0.84%	997.29
	F20n2	0.00%	0.46	F50n2	4.12%	33.70	F100n2	0.31%	828.74
	F20n3	0.00%	0.42	F50n3	0.00%	20.18	F100n3	0.11%	861.88
	F20n4	0.00%	0.35	F50n4	0.00%	19.43	F100n4	0.06%	763.88
	F20n5	0.35%	0.46	F50n5	0.09%	20.29	F100n5	0.41%	1308.33

Table 3.7 shows the average improvement of the (S_{heur}) relaxation over (S_{base}) in each problem type/size category, as well as the number of problems in each category for which the (S_{heur}) relaxation is nearly exact, quantified as having a relative optimality gap of less than 0.005%. The numbers in parentheses indicate the number of problems in a given category for which (S_{base}) was already nearly exact. We observe that the aggregated constraint has a greater impact on the nonconvex Type F problems. Eight problems have exact relaxations using the (S_{heur}) formulation, as opposed to three using the (S_{base}) formulation.

Table 3.7 Average relative gap improvement by using relaxation (S_{heur}) instead of relaxation (S_{base}).

	Average gap_S% improvement		Nearly exact relaxations	
	Type 'B'	Type 'F'	Type 'B'	Type 'F'
$N = 20$	0.12%	0.23%	1 (0)	4 (2)
$N = 50$	0.08%	0.18%	0 (0)	2 (1)
$N = 100$	0.08%	0.57%	1 (0)	0 (0)

3.5 An iterative approach

Although (S_{heur}) gives a significant improvement over (S_{base}) with minimal impact on problem size, we may suspect that there is still room for improvement. In this section we will outline an iterative approach which begins with a relatively low cost model and then strengthens the model by iteratively identifying violated constraints and adding them to the model. The goal of this approach is to capture that the tightness of (S_{full}) with relatively few constraints.

Algorithm overview

The outline of the iterative procedure we will follow is as follows:

1. **Initialize:** Let $k = 0$. Formulate an initial relaxation (S^0_{iter}) and solve it to obtain solution X^*_0.
2. **Score cuts:** At X^*_k, compute violation scores V^{SA}_k and V^{enh}_k for potential Sherali-Adams cuts and enhanced equality cuts, respectively.
3. **Normalize scores:** Normalize cut scores in an appropriate manner.
4. **Rank cuts:** Within each cut type, rank potential cuts by normalized violation score.
5. **Filter cuts:** Using desired criteria, choose the sets of cuts $\mathbb{C}_{SA}$ and $\mathbb{C}_{enh}$ to be added to the model this iteration.
6. **Stop or update:** STOP if $|\mathbb{C}_{SA}| + |\mathbb{C}_{enh}| = 0$ or if another desired stopping criteria is reached. Otherwise let $k = k + 1$, add the cuts of $\mathbb{C}_{SA}$ and $\mathbb{C}_{enh}$ to (S^{k-1}_{iter}) to make (S^k_{iter}), solve to obtain the new solution X^*_0, and return to step 2.

We will now discuss each step in greater detail.

Initialize:

Let $k = 0$. Formulate and solve (S_{heur}) or another known practicably solvable relaxation following the general paradigm discussed in section 3.1.1. Call this relaxation (S^0_{iter}), and denote its SDP solution matrix X^*_0.

Score cuts:

Form the matrices $V^k_{SA} = -\begin{bmatrix} h & -G \end{bmatrix} X^*_k \begin{bmatrix} h^T \\ -G^T \end{bmatrix}$ and $V^k_{enh} = \left| \begin{bmatrix} b & -A \end{bmatrix} (X^*_k) \right|$.

Element $[V_k^{SA}]_{ij}$ corresponds, if positive, to the violation of the valid Sherali-Adams constraint constructed using inequality constraints $i \in I$ and $j \in I$, while $[V_k^{enh}]_{ij}$, corresponds to the violation of the valid enhanced equality constraint constructed by multiplying equality constraint $i \in E$ by variable x_j, $j \in \{1, \dots n\}$.

For convenience, we will mask these matrices to ensure that certain cuts are disregarded:

1) $[V_k^{SA}]_{ij} \leftarrow \begin{cases} [V_k^{SA}]_{ij}, & \text{if } [V_k^{SA}]_{ij} \geq tol \\ 0, & \text{otherwise} \end{cases}$ for all $i, j \in I$.

2) $[V_k^{enh}]_{ij} \leftarrow \begin{cases} [V_k^{enh}]_{ij}, & \text{if } [V_k^{enh}]_{ij} \geq tol \\ 0, & \text{otherwise} \end{cases}$ for all $i \in E, j \in \{1, \dots, n\}$.

3) $[V_k^{SA}]_{ij} \leftarrow 0$ if an (i, j) complementarity constraint is imposed.

4) $[V_k^{SA}]_{ij} \leftarrow 0$ if an (i, j) Sherali-Adams constraint is imposed.

5) $[V_k^{enh}]_{ij} \leftarrow 0$ if an enhanced equality constraint exists between equality i and x_j.

The first two adjustments serve to ignore any violations less than some tolerance *tol*. The latter three make sure that a constraint will be given a score of zero if it (or something stronger than it, in the case of (3)) is already part of the model, which ensures that we won't re-add a constraint which is already part of the model but happened to be violated at X_k^* for any reason.

Normalize scores:

We now normalize the scores to account for scaling differences between constraints.

1) Let $[V_k^{SA}]_{ij} \leftarrow \frac{[V_k^{SA}]_{ij}}{\left\| \frac{h_i}{g_i} \right\|_2 \left\| \frac{h_j}{g_j} \right\|_2}$ for all $i, j \in I$.

2) Let $[V_k^{enh}]_{ij} \leftarrow \frac{[V_k^{enh}]_{ij}}{\left\| \frac{b_i}{a_i} \right\|_2}$ for all $i \in E, j \in \{1, \dots, n\}$.

Rank cuts:

Create a list L_{SA} populated with tuples $([V_k^{SA}]_{ij}, i, j)$ in descending order by the values of their first elements, violation scores $[V_k^{SA}]_{ij}$. Each tuple contains the information for a Sherali-Adams constraint: the size of its violation score at X_k^*, and the inequality indices i and j which correspond to it. Note that L_{SA} need only contain those elements with $i < j$ due to the symmetric nature of Sherali-Adams constraints, and both lists should omit those items with 0 violation.

Similarly, create a list L_{enh} of tuples $([V_k^{enh}]_{ij}, i, j)$ in descending order by violation score $[V_k^{enh}]_{ij}$. Again, each tuple contains the relevant information about an enhanced equality constraint: its violation score and the corresponding equality constraint index i and variable index j.

Filter cuts:

In this step, we choose which constraints will be added to the SDP relaxation for the next iteration. This is done by filtering the list according to any number of rules. We have in fact already pre-emptively applied a few common sense filters to the list by fixing scores for existing constraints and constraints violated within tolerance to zero during the scoring step, ensuring that they will not be chosen. Some other rules one might imagine are:

1) At most K^{SA} Sherali-Adams cuts in total may be added per iteration.

2) At most K^{enh} enhanced equality cuts may be added per iteration.

3) Each inequality $i \in I$ may be involved in at most K_{ineq}^{SA} Sherali-Adams cuts per iteration.

4) Each equality $i \in E$ may be involved in at most K_{eq}^{enh} enhanced equality cuts per iteration.

5) At most K_x^{enh} enhanced equality cuts based on a particular variable x_j may be added per iteration.

6) Consecutive dropoff: In either L_{SA} or L_{enh}, for any two consecutive tuples (v_c, i_c, j_c) and (v_d, i_d, j_d) such that $\alpha v_c > v_d$ (for some chosen $\alpha \in [0, 1]$), the latter cut and all lesser violated cuts are ineligible to be chosen in that iteration.

7) Global dropoff: Letting v_0 be the largest violation score in a given list L_{SA} or L_{enh} at a particular iteration, no cuts with violation scores less than threshold βv_0 may be chosen in that iteration, for some chosen $\beta \in [0, 1]$.

Let $\mathbb{C}_{SA}$ and $\mathbb{C}_{enh}$, initialized as empty sets, denote the sets of cuts which are to be added in the next SDP iteration. We can first truncate the tails of L_{SA} and L_{enh} if cutoff rules such as (6) or (7) are being used. The remaining potential cuts are then considered in most violated order and are chosen to be implemented if they can be added to $\mathbb{C}_{SA}$ and $\mathbb{C}_{enh}$ without breaking any of the remaining rules.

Stop or update:

If $\mathbb{C}_{SA} = \mathbb{C}_{enh} = \emptyset$, no cuts have been added and the algorithm terminates. Otherwise, increment k, add the selected cuts to the SDP model to produce (S_k^{iter}), solve it, and return to the **Score cuts** step.

Notes about filtering rules

Theoretically, if *tol* is set to 0, the optimal value of (S^k_{iter}) will converge to that of S_{full} after a finite number of iterations. In practice, however, it is advisable to use a small a nonzero tolerance *tol* to avoid adding those cuts which may have received a barely-positive score due to numerical error in the interior point solver which produces X^* matrices which are not perfectly feasible for the problems they claim to solve.

The parameters $K^{SA}, K^{enh}, K^{SA}_{ineq}, K^{enh}_{eq}, K^{enh}_{x}$ play an important role in the tuning of the iterative method. Each of these parameters limits the number of constraints chosen from a particular group of constraints in the same iteration. The reason for this is that many Sherali-Adams constraints are associated with the same inequality constraint of the original problem (*P*), or many enhanced equality constraints are associated with the same equality constraint or variable x_j. If (S^k_{iter}) has some exploitable looseness such that several of the highest scoring cuts share a common index, it is likely that adding a small number of cuts from that group will be enough to remove the potential for exploitation. It is also important that we be judicious with the addition of cuts since we are not removing cuts in this method, to avoid cycling.

Evaluating the iterative method

We will test the iterative method for the same problem set introduced in section 3.1.4, and with the parameters listed in Table 3.8.

Table 3.8 Parameters used to test the iterative semidefinite relaxation method

	Test method name		
	SDP_{heur}	$SDP_{heurlim}$	$SDP_{baselim}$
Initial SDP formulation	(S_{heur})		(S_{base})
K^{SA}_{ineq}	–	3	
K^{enh}_{eq}	–	3	
K^{enh}_{x}	–	3	
K^{SA}	50		
K^{enh}	40		
tol	10^{-3}		
α	0.2		
β	–		

The first two iterative methods begin with the relaxation formulation (S_{heur}), with the first method

not placing limits on the number of index sharing constraints which can be chosen (filtering rules (3)-(5)) while the second method does. For comparison, the third iterative method tested begins with (S_{base}), and also enacts filtering rules (3)-(5).

Figure 3.4 shows how the SDP bounds progress over the course of four iterations for the type B, size 100 problems. Remarkably, we find that the *SDPheur* and *SDPheurlim* test methods were almost never able to find cuts which further improved the bound. That does not mean that the iterative method is failing, however. In fact, it is evidence that the initial relaxation (S_{heur}) is already as tight (in terms of objective value) as (S_{full}), despite only having the base model strengthened with the single aggregated equality constraint. Equally remarkably, no equality constraints were added at any iteration for *SDPheur* and *SDPheurlim*, the methods which used the aggregated equality constraint. Then for these problems, only valid Sherali-Adams cuts were identified and added. Although the bound did not change, it is possible that X^* at one iteration could be better suited to a particular purpose than X^* at another iteration. For example, we might hypothesize that X^* becomes closer to rank one in later iterations, a notion we will talk more about in the next chapter.

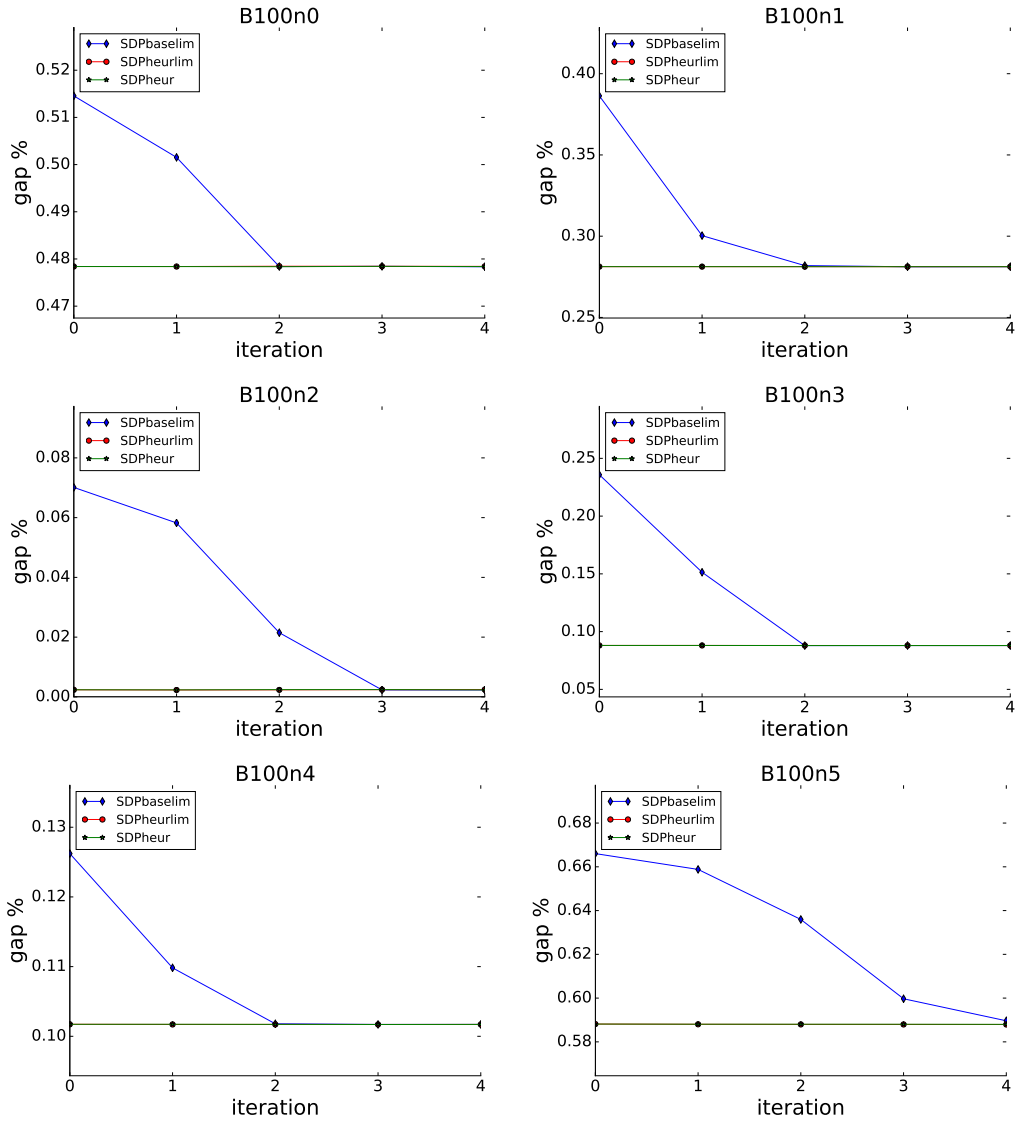

Figure 3.4 Progress in gap for the iterative method with three sets of parameters.

Figure 3.5 shows something not seen in Figure 3.4, which is an example where SDP_{heur} and $SDP_{heurlim}$ are distinguishable. For this problem we find that $SDP_{heurlim}$ and even eventually $SDP_{baselim}$ drop slightly below the line set by SDP_{heur}. This speaks to the notion that it may be possible to achieve more tightness with fewer constraints when filtering rules (3)-(5) are used to ensure that a diverse range of cuts are added at each iteration.

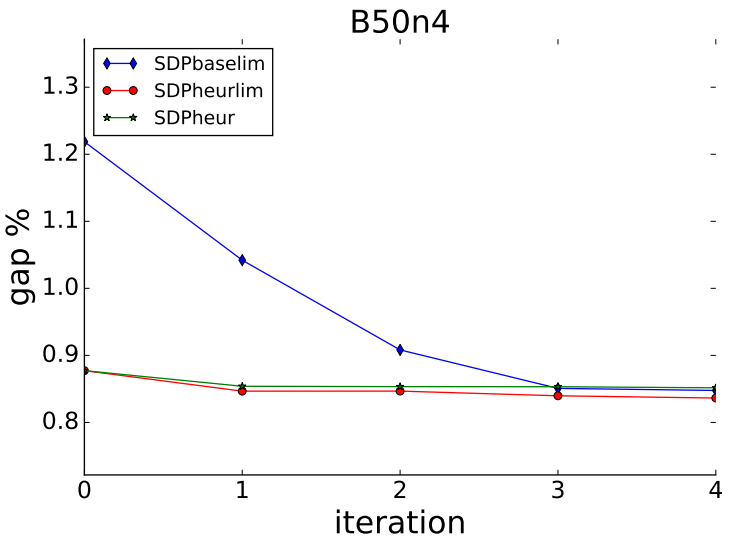

Figure 3.5 For problem B50n4, we see $SDP_{heurlim}$, and eventually even $SDP_{baselim}$, undercut SDP_{heur} by a small amount.

CHAPTER 4 CANDIDATE POINT BASED HEURISTIC METHODS FOR QPLCCS

It is common knowledge among practitioners of semidefinite programming that a rank one optimal solution X^* to a relaxation (S) of the form introduced in 1.1.4 can be mapped back to the space of the original problem (P) to yield an optimal solution x^*. We formalized this for our SDP relaxations of the QPLCC with Theorem 3.1.2, Corollary 3.1.3, and Corollary 3.1.4, with extensions to relaxations with other valid tightening constraints as discussed in section 3.3.

However, in practice it is unlikely that solving an SDP relaxation of a continuous problem such as a QPLCC will yield a perfectly rank one solution X^*, if for no other reason than the numerical error and acceptable feasibility tolerances inherent in interior point solution algorithms. In this chapter, we extend the notion of mapping solutions to (S) back to the space of (P), proposing a number of procedures to map an SDP relaxation solution X^* of arbitrary rank to a point in the space of the original problem, which we call a candidate point. We will define a candidate point as a point which is found heuristically and proposed as an estimator of a global solution to (P), without necessarily any guarantees of feasibility, optimal value, or proximity to a true optimal solution. The procedure is not specific to relaxations of QPLCCs and can be applied to any relaxations of the nature described in 1.1.4.

For the same QPLCC problems used experimentally in chapter 3, we perform computational tests to support discussion of the candidate point's suitability as an estimator of a global solution. We also discuss questions relating to the potential use of the candidate point in procedures for assisting local or global solvers by warmstarting.

4.1 Rank one X^*

Before we define possible candidate points, we will briefly recall how an SDP relaxation's rank one solution X^* can be mapped back to the space of the original problem (P). Let X^* be an optimal solution to an SDP relaxation constructed using the procedure given in section 1.1.4, such as any of the relaxations detailed in chapter 3. Recall that a vector x^* can be extracted from the matrix X^* as visualized in (4.1):

$$X^* = \begin{pmatrix} X_{00} & X_{01} & \cdots & X_{0n} \\ \boxed{X_{10}} & \ddots & & \vdots \\ \boxed{\vdots} & & \ddots & \vdots \\ \boxed{X_{n0}} & \cdots & \cdots & X_{nn} \end{pmatrix}, \qquad x^* = \boxed{\begin{matrix} X_{10} \\ \vdots \\ X_{n0} \end{matrix}}. \tag{4.1}$$

4.2 Higher rank X^*

Recall that a symmetric matrix X has the singular value decomposition

$$X = \sum_{i=1}^{r} \lambda_i q_i q_i^T \tag{4.2}$$

for nonnegative scalars $\lambda_1, \dots \lambda_r$ and orthonormal vectors $q_1, \dots q_r$, where r is the rank of X. Without loss of generality, we will assume that the components are ordered such that $\lambda_i \geq \lambda_j \; \forall \; i = 1, \dots, r, \; j = 1, \dots, r, \; i < j$.

It is known that the rank-k approximation X^k of X^* which minimizes approximation error as measured by either $\|X^* - X^k\|_2$ or $\|X^* - X^k\|_F$ is given by truncating the sum in (4.2) to keep only the components with the k largest λ_i values, i.e.

$$X^k = \sum_{i=1}^{k} \lambda_i q_i q_i^T \tag{4.3}$$

(Stewart, 1993).

Furthermore, since q_i are unit vectors, the magnitude of each component is tied directly to the eigenvalues λ_i which serve as weights, with

$$\|\lambda_i q_i q_i^T\|_2 \;\; = \;\; \sup_{\|y\|_2=1} \|\lambda_i q_i q_i^T y\|_2 \;\; \leq \;\; |\lambda_i| \, \|q_i\|_2 \sup_{\|y\|_2=1} q_i^T y \;\; \leq \;\; \lambda_i \max_{j=1,\dots,n} |q_i|_j \;\; \leq \;\; \lambda_i \;\; . \tag{4.4}$$

Then, in approximating X^* by the rank-one matrix $X^1 = \lambda_1 q_1 q_1^T$, the 2-norm of the error can be bounded as follows:

$$\|X^* - X^1\|_2 \;\; = \;\; \| \sum_{i=2}^{r} \lambda_i q_i q_i^T \|_2 \;\; \leq \;\; \sum_{i=2}^{r} \lambda_i \;\; . \tag{4.5}$$

With this in mind, we will define a metric $R(X)$ to gauge how 'close' a matrix X is to being rank one. We will define $R(X)$ to be

$$R(X) = \frac{\sum_{i=2}^{r} \lambda_i(X)}{\sum_{i=1}^{r} \lambda_i(X)} \;\; , \tag{4.6}$$

where $\lambda_i(X^*)$ is the i^{th} largest eigenvalue of the relaxation's optimal solution X. The R score indicates the proportion of the variation in X which is not captured by its rank one approximation $\lambda_1 q_1 q_1^T$. The R score will always lie in $[0, 1)$, with a score of 0 meaning that X is rank one.

We will say that X is close to rank one if $R(X)$ is near zero since this means that, for rank one

approximation $X^1 = \lambda_1 q_1 q_1^T$, the 2-norm of the error matrix $\|X - X^1\|_2$ is bounded to be small relative to the sum of all eigenvalues of X, also known as the trace norm.

Having observed in chapter 3 that (S_{heur}) yields good bounds for many test problems, let us consider the SDP solutions X given by this relaxation with respect to the R metric. Figure 4.1 demonstrates the distribution of $R(X^*)$ versus relative gap when using two iterative methods detailed in Table 3.8, both after the initial solve and after four iterations of adding tightening cuts and resolving.

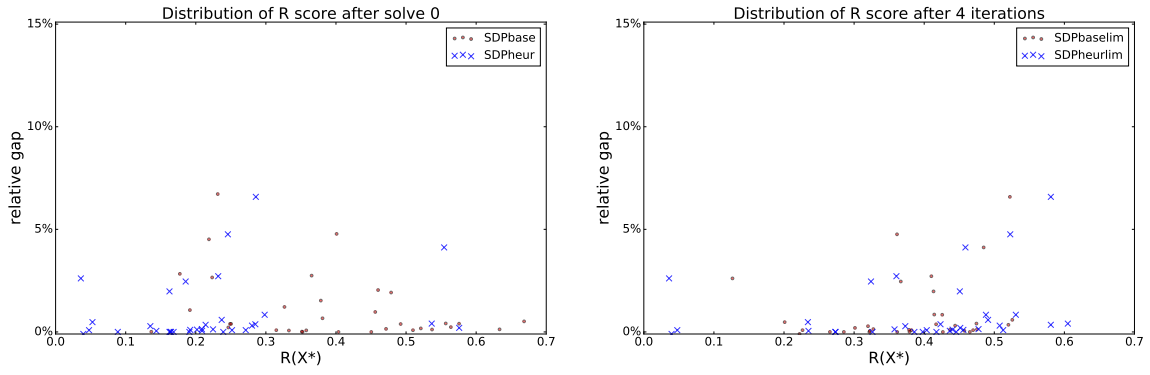

Figure 4.1 Distribution of $R(X^*)$ for solutions of the iterative method.

What can be observed from Figure 4.1 is that initial solves for (S_{heur}) tend to give solutions X^* which are closer to rank 1 than the solutions for (S_{full}), and that, contrary to our expectation, neither iterative method $SDP_{baselim}$ or $SDP_{heurlim}$ can positively be said to be producing solutions which are closer to rank one as the iterative method progresses.

4.3 Candidate point terminology and motivation

In this chapter we are interested in using the solution to an SDP relaxation to compute a candidate point. We have chosen the term candidate point deliberately in order to express two things:

1. The point is a candidate to be a global optimal solution, in the sense that is the result of a heuristic attempt to approximate such.

2. A candidate point is not necessarily feasible, and so to prevent confusion we must not term it a 'solution' of any kind.

In other words, the term 'candidate point' does not in and of itself imply that that the point will meet any particular criteria, but rather serves as a label indicating the role in which a user intends

to use the point in a larger method, namely as an approximation of or proxy for the global optimal solution.

By this definition, we consider that there are three arenas in which a candidate point might be evaluated:

1. Evaluating the candidate point in the objective function of the original problem, how close is the result to the original problem's global optimal value?
2. Evaluating the candidate point in the constraints of the original problem, how big are the constraint violations?
3. Calculating the relative distance between the candidate point and the closest global optimal solution, how close are they in space?

The question of what makes one candidate point better than another depends on how one prioritizes these arenas, as well as on the context of the larger method in which the candidate point will be used. For example, for some purposes an infeasible candidate which hints at the neighbourhood of a global optimum may be preferable to a candidate point which is feasible but far from the global optimum, and vice versa.

We will now introduce four possible recipes for deriving candidate points from X^* with arbitrary rank.

Linear proxy candidate point

The linear proxy candidate point is named as such because it is obtained by constructing a vector which takes its values from the terms of X^* which served as proxies for the linear terms x in the $(P) \rightarrow (S)$ lifting. In other words, candidate point $\hat{x}$ is extracted from X^* in the same way that a global optimal solution would be extracted if X^* were rank one:

$$X^* = \begin{pmatrix} X_{00} & X_{01} & \cdots & X_{0n} \\ X_{10} & X_{11} & & \vdots \\ \vdots & & \ddots & \vdots \\ X_{n0} & \cdots & \cdots & X_{nn} \end{pmatrix}, \qquad \hat{x} = \begin{bmatrix} X_{10} \\ \vdots \\ X_{n0} \end{bmatrix}. \tag{4.7}$$

One advantage of this candidate point is that the linear equality and inequality constraints of the original QPLCC are enforced directly on these linear proxy variables of X^*, so the linear constraints will be satisfied by this candidate point. On the other hand, the complementarity constraints may be

violated and the objective value at the candidate point may be very different from the SDP bound given by X^*.

Square proxy candidate point

The square proxy candidate point is similar to the linear proxy candidate point in that it is also based on the assumption that particular elements of the matrix X^* are accurate proxies for the quantities they were intended to serve as proxies for. In this square case, the assumption which is relied upon is that X_{ii} is an accurate proxy for the term x_i^2, and so element i of the candidate point $\hat{x}$ takes its magnitude from the square root of the diagonal element and its sign from the value of X_{i0}, ie.

$$X^* = \begin{pmatrix} X_{00} & X_{01} & \cdots & X_{0n} \\ X_{10} & X_{11} & & \vdots \\ \vdots & & \ddots & \vdots \\ X_{n0} & \cdots & \cdots & X_{nn} \end{pmatrix}, \qquad \hat{x} = \begin{pmatrix} s_1\sqrt{X_{11}} \\ \vdots \\ s_n\sqrt{X_{nn}} \end{pmatrix}, \tag{4.8}$$

where

$$s_i = \begin{cases} \frac{X_{i0}}{|X_{i0}|} & X_{i0} \neq 0 \\ 0 & X_{i0} = 0 \end{cases}. \tag{4.9}$$

Rank one approximation candidate point

While the linear proxy candidate point can be summarized as treating X^* the same way we would if it were rank one, the rank one approximation candidate point is computed by first approximating X^* with a rank one matrix X^1 and then extracting $\hat{x}$ from X^1 in the usual manner:

$$X = \sum_{i=1}^{r} \lambda_i q_i q_i^T, \tag{4.10}$$

$$X^1 = \lambda_1 q_1 q_1^T = \begin{pmatrix} X^1_{00} & X^1_{01} & \cdots & X^1_{0n} \\ X^1_{10} & X^1_{11} & & \vdots \\ \vdots & & \ddots & \vdots \\ X^1_{n0} & \cdots & \cdots & X^1_{nn} \end{pmatrix}, \qquad \hat{x} = \begin{pmatrix} X_{10} \\ \vdots \\ X_{n0} \end{pmatrix} = \begin{pmatrix} \lambda_1(q_1)_0 \\ \lambda_1(q_1)_1 \\ \vdots \\ \lambda_1(q_1)_n \end{pmatrix}. \tag{4.11}$$

For this candidate point, we can make the claim that when $R(X^*)$ is very small, the candidate point will be ‘almost feasible’ in the sense that each constraint will be violated by at most a small amount.

For example, the violation of an equality constraint $a_i^T \hat{x} = b_i$ can be shown to be bounded above by

$$|b - a^T \hat{x}| \leq \frac{1}{s_1 \sqrt{\lambda_1}} \left\| \begin{matrix} b_i \\ -a_i \end{matrix} \right\|_2 \sum_{k=2}^{r} \lambda_k \quad . \tag{4.12}$$

Adjusted rank one approximation candidate point

This variation is motivated by an example which highlights a potential flaw of the rank one approximation approach.

Example 4.3.1. Consider the simple problem (P_0) and associated SDP relaxation (S_0):

$$\begin{array}{lll} \min\ y = x^2 & & \min\ y \\ \text{s.t. } x^2 = 1 \quad (P_{Ex}^4) \ , & & \text{s.t. } y = 1 \qquad\qquad (S_{Ex}^4) \\ & & X = \begin{bmatrix} 1 & x \\ x & y \end{bmatrix} \geq 0 \ . \end{array} \tag{4.13}$$

(P_0) has exactly two feasible solutions, $x = -1$ and $x = 1$, and both are optimal. (S_0) is an exact SDP relaxation of (P_0), but its set of optimal solutions can be expressed parametrically as

$$Sol(S_0) = \left\{ \begin{bmatrix} 1 & \theta \\ \theta & 1 \end{bmatrix}, \ \theta \in [-1, 1] \right\} \quad , \tag{4.14}$$

which is the convex hull of the two rank one solutions $\left[\begin{smallmatrix} 1 & -1 \\ -1 & 1 \end{smallmatrix}\right]$ and $\left[\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right]$, corresponding to $x = -1$ and $x = 1$, respectively. Supposing we solve the model with an interior point solver, we may obtain a solution which is a convex combination of two rank one optimal solutions. For example, the solution and its singular value decomposition may be as follows:

$$\begin{aligned} X^* &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ &= \lambda_1 q_1 q_1^T + \lambda_2 q_2 q_2^T \quad , \\ &\quad \text{where } \lambda_1 = 0.5\sqrt{2}, \quad q_1 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} \end{bmatrix}, \quad \lambda_2 = 0.5\sqrt{2}, \quad q_2 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \quad , \end{aligned} \tag{4.15}$$

leading to the choice of either $x \approx 0.707$ or $x \approx -0.707$, neither of which is feasible for (P).

We can propose an adjustment step which ensures that the candidate point computed from a given rank one matrix $\lambda_i q_i q_i^T$ is the same regardless of the weight which is placed on it, allowing a true

global solution to be extracted from X^*. Partitioning the vector q_1 into $\begin{bmatrix} s_1 \\ \bar{q}_1 \end{bmatrix}$, where s_1 is the first element and $\bar{q}_1$ is a vector with the remaining elements, X^1 can be rewritten as

$$X^1 = \lambda_1 q_1 q_1^T = \lambda_1 \begin{bmatrix} s_1 \\ \bar{q}_1 \end{bmatrix} \begin{bmatrix} s_1 \\ \bar{q}_1 \end{bmatrix}^T = \lambda_1 \begin{bmatrix} s_1^2 & s_1 \bar{q}_1^T \\ s_1 \bar{q}_1 & \bar{q}_1 \bar{q}_1^T \end{bmatrix} \quad . \tag{4.16}$$

Since each q_i is a vector of unit length, λ_i and s_i play a weighting role in the decomposition of X^* as a sum of rank one matrices. To compute the adjusted rank one approximation candidate point we will select the component which has the largest λ value, scale the corresponding matrix X^1 so the first element is equal to 1, and then proceed to extract a candidate point as in the rank one approximation case:

$$\hat{X} = \begin{bmatrix} 1 & \frac{1}{s_1} \bar{q}_1^T \\ \frac{1}{s_1} \bar{q}_1 & \frac{1}{s_1^2} \bar{q}_1 \bar{q}_1^T \end{bmatrix} \quad , \quad \hat{x} = \frac{1}{s_1} \bar{q}_1 \quad . \tag{4.17}$$

By this method, the candidate point extracted from X^* for (S) will be one of the optimal solutions $x = -1$ or $x = 1$, as desired.

4.4 Comparing candidate points

In this subsection we will examine the candidate points which can be computed from each test problem's SDP relaxation solution X^*. For these results we will focus on the case of the iterative SDP relaxation method, initialized from (S_{heur}). X^* is taken either at termination or after 5 outer iterations.

4.4.1 Introducing metrics

We will define three types of metrics which we will use to evaluate and compare the four different candidate point definitions. The metrics correspond to the three qualities discussed in section 4.3: objective value, feasibility, and proximity to a global optimal solution.

Gap

To measure how well $\hat{x}$ reflects an optimal solution x with regards to objective value, we will let gap_c be the absolute value of the relative gap between the optimal value z^* and the value of the

objective function evaluated at $\hat{x}$.

$$gap_c^* = |\frac{z(\hat{x}) - z^*}{z^*}| \ , \tag{4.18}$$

Alternatively, we may want a metric to evaluate the objective value achieved by $\hat{x}$ which does not use the global optimal value, so we may want to use these metrics to assess a problem which has not been globally solved. In this case, we can define the alternative metric,

$$gap_c^S = |\frac{z(\hat{x}) - z_S(X^*)}{z_S(X^*)}| \ . \tag{4.19}$$

The interpretation of gap_c^S is that it measures how close $\hat{x}$ comes to matching its associated bound value $z_S(X^*)$.

Feasibility

The second metric of interest is a measure of the infeasibility of $\hat{x}$ for the QPLCC, calculated as

$$viol_c = \frac{1}{|E|}\sum_{i\in E}|\tilde{f}_i^T\hat{x} - \tilde{h}_i| + \frac{1}{|I|}\sum_{i\in I}\max(0,\ \tilde{a}_i^T\hat{x} - \tilde{b}_i) + \frac{1}{|C|}\sum_{(i,j)\in C}|(\tilde{b}_i - \tilde{a}_i^T\hat{x})(\tilde{b}_j - \tilde{a}_j^T\hat{x})| \ , \tag{4.20}$$

where $\tilde{f}_i = \alpha_i f_i, \quad \tilde{h}_i = \alpha_i h_i, \quad \alpha = \left\| \begin{smallmatrix} h_i \\ f_i \end{smallmatrix} \right\|_2^{-1}$, and $\tilde{a}_i = \beta_i a_i, \quad \tilde{b}_i = \beta_i b_i, \quad \beta = \left\| \begin{smallmatrix} b_i \\ a_i \end{smallmatrix} \right\|_2^{-1}$.

This corresponds to the sum of the average normalized equality constraint violation, average normalized inequality constraint violation, and average normalized complementarity constraint violation, providing a metric of overall feasibility.

Proximity to a global solution

The third and final metric which we will introduce for examining a candidate point is its 'relative distance'. Since we know the global solution x^* to each problem in the test set, we can compute the distance from the candidate point to the global solution relative to the 2-norm of x^*:

$$dist_c = \frac{||\hat{x} - x^*||_2}{||x^*||_2} \ . \tag{4.21}$$

4.4.2 Evaluating candidate points

For each of the problems introduced in section 3.1.4, we now use the metrics defined in the last section to score various candidate points derived from various semidefinite relaxations. Excerpts are presented in this section, and full tables for two relaxations and all four candidate points can be

seen in Appendix A and B. Each candidate point is evaluated using the metrics defined in section 4.4.1: $gap_c(\hat{x})$, $viol_c(\hat{x})$, and $dist_c(\hat{x})$, with the SDP solution's optimality gap and $R(X^*)$ measure provided for reference. In the tables to come, LP, SP, R1, and AR1 stand for the linear proxy, square proxy, rank one, and adjusted rank one candidate points, respectively.

Table 4.1 Evaluating candidate points derived from the solution to (S_{heur}).

	SDP sol. X^*		candidate point $\hat{x}$			
Name	$gap_S(X^*)$ %	$R(X^*)$	type	$gap_c(\hat{x})$	$viol_c(\hat{x})$	$dist_c(\hat{x})$
B100n0	0.48%	0.05	LP	0.48%	0.00	20.51%
			SP	0.48%	0.38	28.85%
			R1	0.48%	0.00	20.51%
			AR1	0.48%	0.00	20.51%
B100n1	0.28%	0.14	LP	0.28%	0.00	24.29%
			SP	0.28%	4.10	40.80%
			R1	0.28%	0.01	24.28%
			AR1	0.28%	0.01	24.28%
B100n2	0.00%	0.09	LP	0.00%	0.00	8.14%
			SP	0.00%	0.10	31.79%
			R1	0.00%	0.00	8.14%
			AR1	0.00%	0.00	8.14%
B100n3	0.09%	0.27	LP	0.09%	0.00	5.83%
			SP	0.09%	8.20	58.83%
			R1	0.09%	0.01	5.84%
			AR1	0.09%	0.01	5.84%
B100n4	0.10%	0.19	LP	0.10%	0.00	10.39%
			SP	0.10%	4.77	45.08%
			R1	0.10%	0.00	10.37%
			AR1	0.10%	0.00	10.37%
B100n5	0.59%	0.24	LP	0.59%	0.00	29.96%
			SP	0.59%	3.75	56.55%
			R1	0.59%	0.00	29.96%
			AR1	0.59%	0.00	29.96%

Similarity between candidate points

When examining Table 4.1, the first thing we notice is that for almost every problem in the test set, the gap, violation, and distance 'scores' for the linear proxy candidate point are nearly identical to those for the rank one approximation candidate point and the adjusted rank one approximation candidate point, to such an extent that it is reasonable to assume that the three candidate point derivations are nominating almost the same points for these relaxations. However, this is not the case in general, as evidenced by Table 4.2, which shows the same kind of evaluation performed on candidate points derived from the solution to the looser relaxation (S_{base}).

Table 4.2 Evaluating candidate points derived from the solution to (S_{base}).

	SDP sol. X^*		candidate point $\hat{x}$			
Name	$gap_S(X^*)$ %	$R(X^*)$	type	$gap_c(\hat{x})$	$viol_c(\hat{x})$	$dist_c(\hat{x})$
B100n0	0.51%	0.67	LP	0.52%	0.10	21.83%
			SP	0.49%	5.91	237.20%
			R1	99.70%	0.94	100.04%
			AR1	0.01%	54.19	3826.40%
B100n1	0.39%	0.49	LP	0.39%	0.91	30.53%
			SP	0.38%	77.83	765.02%
			R1	99.86%	0.93	100.78%
			AR1	0.93%	402.39	21038.32%
B100n2	0.07%	0.33	LP	0.07%	0.05	13.15%
			SP	0.05%	10.44	1393.76%
			R1	100.00%	0.82	100.09%
			AR1	46.90%	377.84	306163.31%
B100n3	0.24%	0.56	LP	0.24%	0.45	20.57%
			SP	0.23%	55.01	581.52%
			R1	99.65%	0.94	101.11%
			AR1	0.08%	161.19	9322.37%
B100n4	0.13%	0.63	LP	0.13%	0.11	13.35%
			SP	0.13%	15.34	211.25%
			R1	97.38%	0.88	99.96%
			AR1	0.12%	1.00	1239.47%
B100n5	0.67%	0.38	LP	0.67%	0.50	29.45%
			SP	0.66%	76.28	780.45%
			R1	99.99%	0.87	100.09%
			AR1	8.92%	1717.48	75720.92%

We observe that in many cases the rank one approximation candidate point has relatively small violations but a large gap and distance, while the adjusted rank one approximation candidate point often has a reasonable gap score but unreasonable violation and distance scores. On the other hand, the linear proxy candidate point, despite its simplicity, proves to be the most robust candidate point for the relatively loose SDP relaxation (S_{base}).

Feasibility of the linear proxy candidate point

We observed in section 4.3 that the linear proxy candidate point will satisfy the linear constraints of the original problem by design. In practice, we find that for the (S_{heur}) relaxation formulation the linear proxy candidate point also satisfies the complementarity constraints (within a moderate numerical tolerance) for virtually all problems. This is by no means assured to be the case in general; for the looser relaxation (S_{base}) applied to the test problem set, the linear proxy candidate point has nonzero violation score for 35 of 36 problems.

4.5 Methodology for the warmstarting of local and global NLP solvers

We will now turn our attention to a key application of the candidate point, the warmstarting of a local or global solver. We recall that global solver BARON had difficulty solving our test problems with size $N = 100$, particularly the type F problems which have nonconvex quadratic objective functions. Only two of six problems were solved to global optimality within 6 hours, with the fastest taking 6411 seconds. We will consider these problems our target for improvement.

4.5.1 Local NLP solvers

In this section, we discuss how a candidate point computed from an SDP relaxation's solution X^* can be parlayed into a locally optimal solution for the original problem (P). In the best case, X^* is a rank one matrix and therefore the candidate point $\hat{x}$ is in $sol(P)$, the set of globally optimal solutions. In the worst case, there is no guarantee that the candidate point will be within a particular distance of any point in $sol(P)$ or even any feasible solution for (P). It is important to note that the techniques discussed in this section are heuristic and can possibly result in a worse solution than that found by the solver ordinarily.

Local nonlinear programming solvers are a class of NLP solvers which do not pursue global optimality but instead terminate once local optimality is established. These solvers are generally very fast, and can often be easily swayed by the choice of initial solution, meaning that a solver which is initialized very near to a local or global optimum has a good chance to converge to that point. This makes local NLP solvers very compatible with the concept of the candidate point. The solver is initialized at the candidate point, which aims to be an estimator of an global optimal solution. The process of initializing a solver at a point which is believed to be near a good solution is known as warmstarting. If indeed the candidate point is in a small neighbourhood of a very good solution, it is likely that the local solver will find it. On the other hand, a local solver does not require any special conditions and will still be able to function robustly and find some local optimum even if the candidate point is a bad estimator of the global solution(s). We will propose that for purposes

of local warmstarting, the linear proxy candidate point is a good candidate point to use since it is guaranteed to satisfy at least the linear constraints of the problem.

To demonstrate local warmstarting, we will use the local NLP solver KNITRO via the AMPL modeling language. A particular note of interest about the AMPL language is that it has a designated syntax for modeling complementarity constraints. This allows solvers such as KNITRO to implement sophisticated ways of treating complementarity constraints, as opposed to handling them simply as big-M constraints or quadratic equality constraints. Figure 4.2 highlights the effectiveness of warmstarting the KNITRO solver. A scatter plot plots the gaps for KNITRO without a given initial solution against the gaps for KNITRO initialized at the linear proxy candidate point for the (S_{heur}) relaxation formulation. In all but a few cases, the solution by warmstarting is inferior to the solution by default settings. One outlier has been cropped from the graph, at (16.68%, 0%). Furthermore, all warmstarted problems have less than 0.05% gap.

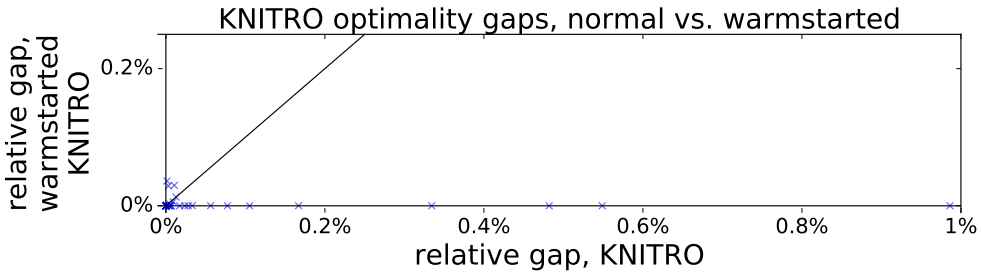

Figure 4.2 For these problems, warmstarting KNITRO at the linear proxy candidate point consistently brings the relative gap very near 0.

4.5.2 Global NLP solvers

We now consider how the methods discussed thus far can be used to assist in the global solution of QPCCs. We will restrict the scope of this investigation, focusing on using the information obtained from an SDP relaxation to assist the well-known global MINLP solver BARON. BARON, an acronym for 'Branch-And-Reduce Optimization Navigator', operates in a branch-and-cut framework, using sophisticated techniques to construct polyhedral cuts for tighter relaxations of nonconvex problems (Tawarmalani and Sahinidis, 2002). Because BARON supports mixed integer problems, there are two options for modeling complementarity constraints. For $(i, j) \in C$, one can write $(h_i - g_i^T x)(h_j - g_j^T x) = 0$. Alternatively, a binary variable z_{ij} can be employed with constraints $h_i - g_i^T x \leq M z_{ij}$ and $h_j - g_j^T x \leq M(1 - z_{ij})$ for sufficiently large M.

So far, we have shown three SDP-derived 'products' which give insight into the original problem (P): a lower bound on the optimal value, a candidate point which aims to estimate a global optimal

solution, and a heuristically found locally optimal solution, assuming the local solver does not stall at a locally infeasible point. It is the result of the local NLP solve which has the greatest potential impact on BARON's solution time.

As in any branching framework, finding a good solution early in the branching tree is highly desirable because it can allow BARON to fathom some later nodes without fully evaluating them. By default, BARON repeatedly applies a multistart approach with a local NLP solver at the root node, which means that it performs local optimization solves beginning from different initial points with the goal of finding a good first incumbent solution before beginning the branch-and-bound procedure. However, a feasible solution which is known to the user can also be provided as an initial solution. We will use this feature to initialize BARON at the known local optima obtained as discussed in the previous section. Results later in this section will compare the time taken to solve with BARON ordinarily to the time taken to solve an SDP relaxation, compute the candidate point, perform a warmstarted local solve, and then perform a warmstarted BARON solve.

SDP bounds and BARON

Studying the log messages of BARON solves, we notice that for several of our larger problems, particularly those with nonconvex objective functions, a better bound is being found by the semidefinite programming relaxation in less time than it takes BARON's process to arrive at a similar bound. For example, for the problem $B100n5$, the (S_{heur}) relaxation provides a bound of -1104.2 after 1308 seconds, whereas BARON has a bound of -1118.31 after roughly the same amount of time and its lower bound reaches only -1108.47 within 6 hours. It is tempting to think that an SDP relaxation's lower bound can be passed to a global solver directly, but this is naive. Unfortunately, at least in the case of BARON, this is more difficult than it appears. Providing a lower bound as either a constraint or a lower bound on the objective value variable is generally counterproductive: in most cases, BARON takes orders of magnitude longer when it has been 'helped' by the addition of an explicit bound. It seems that the organic discovery process by which BARON iteratively tightens its relaxations is necessary for successful branching and navigation of the problem, and the addition of a good bound initially can cripple BARON's ability to branch effectively and build strong relaxations

Candidate points and BARON

BARON allows an initial solution to be provided by the user, but there is a caveat: it must be feasible to a rather high degree of accuracy. It is generally accepted that most interior point solvers for semidefinite programming are typically only accurate to 4 or 5 significant figures, whereas NLP solvers, both local and global, might be accurate to twice that many. As such, even in the case

where an SDP relaxation's solution *should* be rank one, the candidate point which is derived will almost certainly not be feasible, at least not within the tolerances that BARON requires in order to acknowledge a feasible solution.

Warmstarting BARON

Since we cannot warmstart BARON directly from our candidate point, we will instead warmstart BARON by using KNITRO as a processing step between the SDP solve and the BARON solve. KNITRO is warmstarted from a (probably infeasible) candidate point, returning a local solution. This local solution, which is feasible to a degree of numerical accuracy which satisfies BARON, is then given to BARON as an initial solution, i.e. a first incumbent solution and initial upper bound. In each of the graphs of Figures 4.3, the progress of BARON's upper and lower bounds over time is depicted by a grey shaded region. In each case, the red shaded region outlined with a dashed line depicts the progress made by BARON when the warmstarting procedure described in this section is used. The profile for the warmstarted case has been shifted forward an amount of time equivalent to that taken by the SDP and KNITRO solves which the warmstarting technique requires, and a star marks the termination of the warmstarted algorithm, for ease of identification.

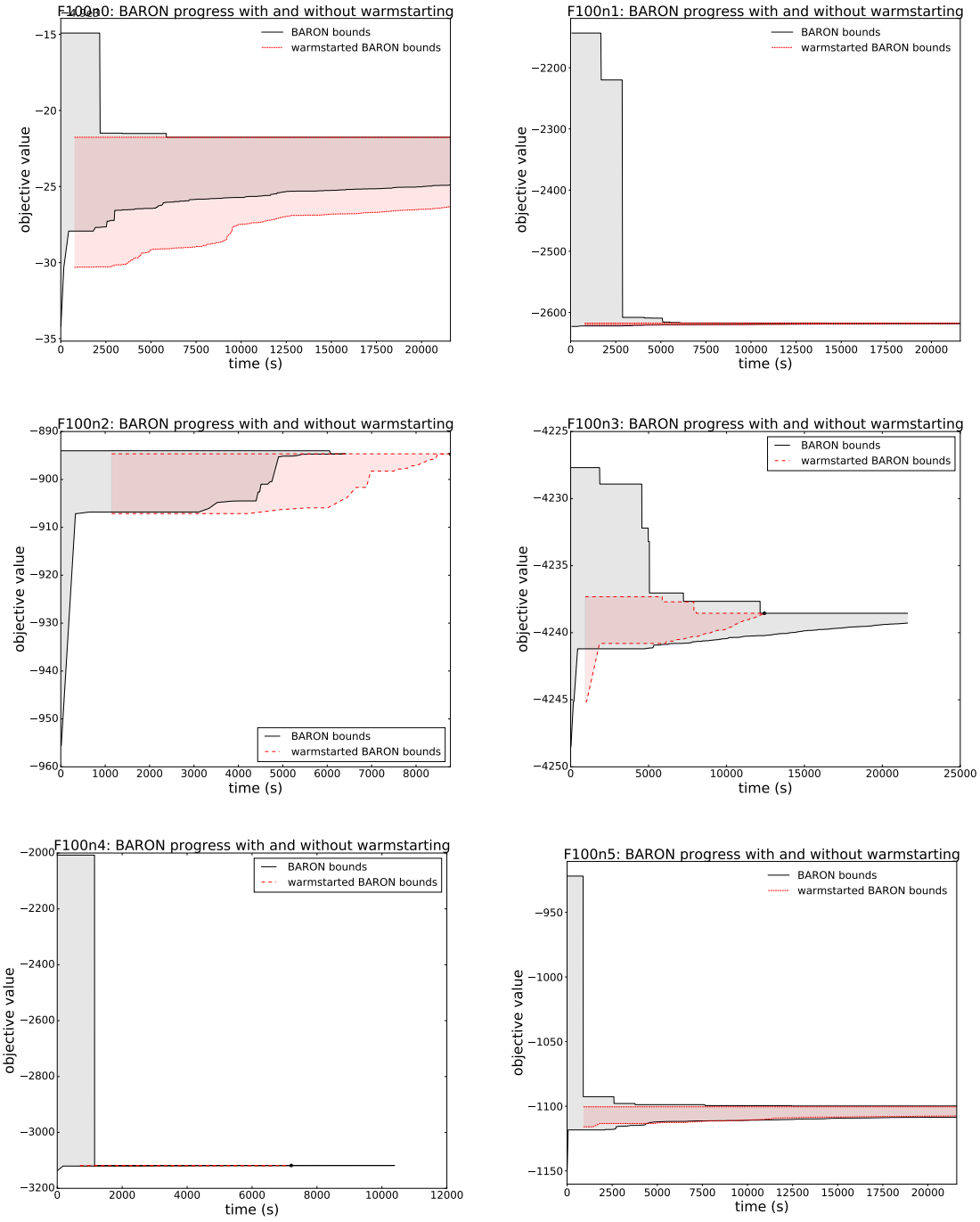

Figure 4.3 Progress made by BARON ordinarily vs. in the warmstarted case

The results of this experiment are mixed. It is known from the gap analysis performed in section 4.5.1 that the warmstart solution provided by KNITRO is globally optimal for two of the problems pictured, F100n2 and F100n4, and so for these problems termination is a matter of BARON's ability

to tighten its relaxations and improve the lower bound so that global optimality can be proven. The warmstarted BARON solve terminates faster for F100n4, but ordinary BARON terminates faster for F100n2. For problems F100n0, F00n1, and F100n5, neither solve terminates within 6 hours, with the warmstarted BARON solve having an inferior profile for problem F100n0 and, in the long run, a slightly better profile for problems F100n1 and F100n5. F100n3 exhibits the behaviour we had hoped to see, with warmstarted BARON having a relatively small initial footprint and a faster termination time.

What is clear from this study is that although SDP provides a stronger initial solution for all six problems, warmstarting BARON at a better initial solution does not guarantee that BARON's overall performance will improve. We attribute this to the nature of BARON, which is that information is accumulated with each iteration and leveraged for better decision making at later iterations. By providing a better initial solution, we deprive BARON of the discovery process that would have been taken to arrive at the same point, and the information produced by that discovery process could be crucial to making good decisions at later iterations.

Evidence for an SDP-based branch and bound

Having observed from the graphs of Figure 4.3 that much of the time required by BARON is to improve the bounds, let us consider how an SDP based branch and bound scheme might compare with BARON's progress. Implementing a full SDP-based branch and bound algorithm is outside the scope of this thesis, but we can think of our SDP bound and KNITRO-provided local optimum as the bounds at the root node and evaluate them as such. In Figures 4.4, SDP-based bounds appear on the graph at the time corresponding to the duration of the SDP solve, and lines are drawn extending the bounds forward in time so that we can visually identify at what time BARON 'catches up' with a given bound. For example, in the case of F100n5, BARON achieves the SDP initial upper bound about 3000 seconds after the SDP achieves it, meanwhile BARON's lower bound has not caught up to the SDP initial lower bound by the 6 hour time limit. Based on this evidence, we believe an SDP-based branch and bound scheme holds promise for the future.

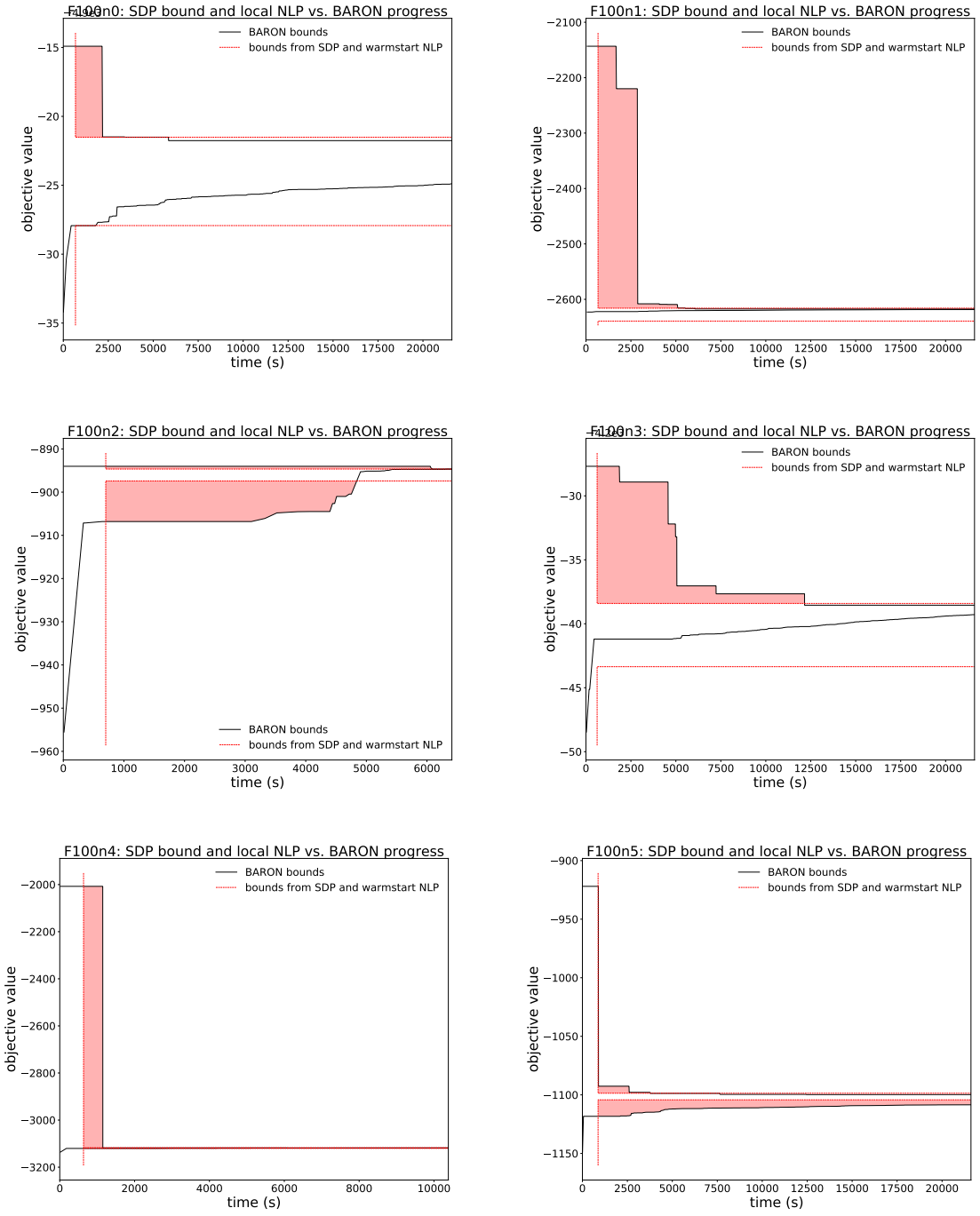

Figure 4.4 The root node SDP bound and warmstarted NLP solution overlaid over BARON's progress

CHAPTER 5 PYTHON IMPLEMENTED SOFTWARE TOOLS FOR QPCCS

Because academics working with QPLCCs employ a wide variety of techniques influenced by a wider variety of disciplines and related problems, there is not (and perhaps cannot be) a consensus about the languages, packages, and other computational tools which are used to support research and development for QPLCCs. This can be detrimental to the field at times, because it creates 'language barriers' between researchers who would like to test each others' methods

In recent years, Python has gained popularity as a language for computational mathematics (Rashed and Ahsan, 2012; Ivezić et al., 2014). As a language, Python strikes an attractive balance between programming generalism, ease of prototyping, and the availability of quality domain-specific scientific computing tools for many fields. Well implemented core numerical packages such as SciPy and NumPy have been embraced wholeheartedly by the scientific community, and individuals working in specific scientific domains have contributed innumerable packages. For example, two Python packages developed for convex optimization and used throughout this thesis are CVXPY (Diamond and Boyd, 2015) and CVXOPT (Vandenberghe, 2010; Andersen et al., 2013).

In this chapter we present three tools based in the Python language which support modeling, solving, and studying quadratic problems with complementarity constraints. Our QPLCC software mission is to create useful tools which reduce these barriers, as well as to integrate a number of tools from various languages into a cohesive platform for research.

PyQPCC, the largest of the three packages we introduce, provides a general framework for modeling QPCCs, managing and analyzing them, translating them to other modeling languages, and building and testing solution methods.

The PyQPECgen package generates test problems according to a number of QPEC types. It is a Python implementation of the Matlab script QPECgen, which is particularly notable for allowing the user control over aspects of interest such as degeneracy in the upper and lower levels (Jiang and Ralph, 1999). Our translation differs from the original in a few ways which are discussed in that section.

Finally, PySDPT3Glue provides an interface for solving Semidefinite, Quadratic, and Linear Programming (SQLP) programs modeled in the Python-based domain-specific language CVXPY using the well known Matlab-based solver SDPT3.

A note of acknowledgement: The three packages discussed in this chapter were originally developed by me alone, but all three have recently gained a second contributor, Junpei Kawamoto of Kyushu University, whose collaboration is much appreciated.

5.1 PyQPCC

Introduction

In addition to being able to meet many scientific needs, one strength of Python is that it is not *solely* a mathematical language. As a generalist language, Python can meet a number of typical programming needs like input/output, regular expressions, subprocess calls, queueing, and multithreading.

As we have mentioned, Python's appeal is as a free open-source language which can meet many scientific needs while also having the high level power to reach outside of itself, using default Python packages to interface with servers or drive other tools via the command line. We use these two methods to build a testing framework which allows complicated experiments using multiple solvers to be quickly prototyped and executed.

PyQPCC provides tools for the modeling and solving QPLCCs as well as the analysis of their results. The package is organized into modules as shown in Figure 5.1. We will discuss the main modules in turn.

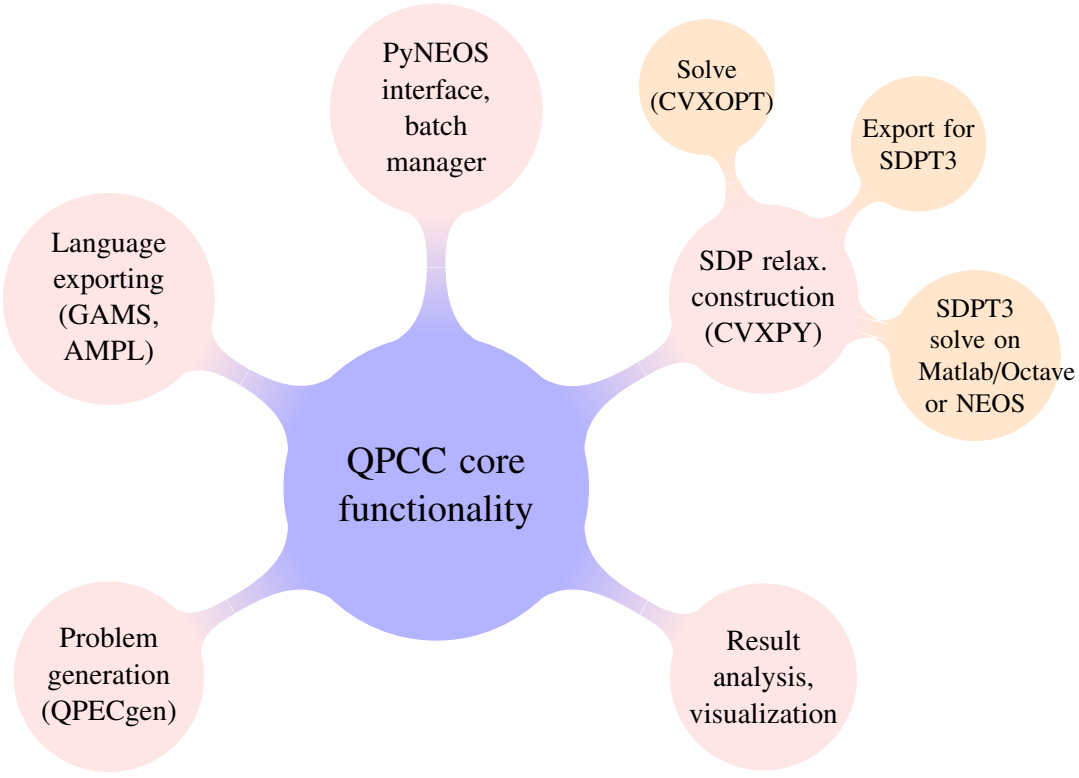

Figure 5.1 A concept map giving an overview of the functionality of PyQPCC

5.1.1 QPLCC modeling

At the core of the PyQPCC package is a module which allows the user to build and manipulate a QPCC model, display it in a user-readable manner, investigate the feasibility of a given point, etc.

A QPCCProblem object contains the data for a problem of the form:

$$
\begin{array}{lll}
\min\limits_{x} & z(x) = x^T Q x + p^T x + r & \\
\text{s.t.} & a_i^T x = b_i & \forall\ i \in E \\
& g_i^T x \leq h_i & \forall\ i \in I \\
& (h_i - g_i^T x)(h_j - g_j^T x) = 0 & \forall\ (i,j) \in C \quad .
\end{array}
\tag{P}
$$

The QPLCC object mirrors this directly. A script to create a toy QPLCC problem might look like the following:

```
from cvxopt import matrix
from pyqpcc.core.qpcc import BasicQPCC

# Initialize the QPCC with a problem name and variable names
Problem = BasicQPCC('qpcc0d', names=['x', 'y'])

# Add equality constraint x+y=0.5 to the model
A = matrix([[1.], [1.]])
b = matrix([0.5])
Problem.add_eqs(A, b)

# make inequality constraints x >= 0, y >= 0, x+y <= 1
G = matrix([[-1., 0., 1.], [0., -1., 1.]])
h = matrix([0., 0., 1.])
Problem.add_ineqs(G, h)

# Require complementarity between the first two equality constraints,
# x >= 0 and y >= 0
comps = [[0, 1]]
Problem.add_comps(comps)

# The objective will be to minize squared distance from (1, 1)
Q = matrix([[1, 0], [0, 1]])
p = matrix([-2, -2], (2, 1))
r = 2.
Problem.set_obj(Q=Q, p=p, r=r, mode='min')
```

5.1.2 Constructing and solving SDP relaxations

In addition to modeling QPLCCs, PyQPCC has a similar object class for modeling semidefinite programs. The SDPProblem class constructs and stores an SDP problem in much the same format as the input format used by solver CVXOPT, so we refer the interested reader to Vandenberghe

(2010) and Andersen et al. (2013). A user can write their own code to construct an SDP relaxation from scratch:

```
# suppose variable P is a QPCCProblem
from pyqpcc.core import sdp
S = sdp.SDPProblem('some_name', size=P.n+1, tolerances=solver_opts['SDP_tols'])

# let's add the aggregated equality for system Ax=b
S.addctr_eqs_squared(self.A, self.b)

# let's impose the equality constraints on the first row of X
for i in range(len(P.b)):
    P.addctr_eq_on_row(P.A, Pb, rowA=i, rowX=0)

(...)
```

or alternatively, some methods exist in the QPCCProblem class which can construct and solve an SDP model according to some preset options:

```
NAIVE_SDP_OPTS = {'SDP_eqs_squared': False, # aggregated equality constraint
                  'SDP_SA_matrix': None,    # Sherali-Adams cuts
                  'SDP_eq_matrix': None}    # Enhanced equality constraints

S = Problem.buildSDP(custom_solver_opts = NAIVE_SDP_OPTS)
result = S.solve(solver='SDPT3')
# let's say we want to know how long the solve took
print result.solve_time
```

Methods also exist for iterative solves following the procedure in section 3.5:

```
ITER_OPTS = {'bundle_SA_per_ineq': None,
             'bundle_eq_per_eq': None,
             'bundle_eq_per_row': None,
             'bundle_K_SA': 50,
             'bundle_K_eq': 40,
             'bundle_drop_cutoff': 0.2,
             'bundle_abs_tol': 10**(-3),
             'bundle_rel_tol': 0,
             'bundle_num_iters': npinf,
             'bundle_global_time_limit': npinf,
             'bundle_solver_time_limit': 21600}

result = Problem.iter_sdp_solve(solver='cvxopt', initial_solver_opts=NAIVE_SDP_OPTS)
# let's say we want the X matrix from the first solve in the sequence:
X = result.sequence[0].X
```

but again, a user can code their own custom version.

5.1.3 Results, result collections and problem series management

Many methods in PyQPCC treat problem series, which are simply lists of QPCCProblem objects. Problem series are convenient for applying the same treatment to a large number of test problems. Additionally, we also provide methods for saving or loading a 'pickled' set of QPCC problems, making it easy to save a set of problems (and results) to disk or load them back into memory.

Each QPCC is initialized with an empty result collection, which is a data structure for storing the results of various solution approaches for that problem. Each time a QPCC problem is solved, a result object is created which holds vital information about the solution method, its parameters, and its outcome. Results store information such as solve time, solver name, status returned by the solver, initial solution (for warmstarted NLP results), and results also have methods such as gap calculators or functions that print summaries of their information. Every result also has two unique identifiers, an ID and a timestamp, which are helpful for extracting particular results as needed.

Results and result collections provide functions that make it easy to analyze results, from basic functions that seek out and retrieve a particular result, to more advanced ones that produce tables and graphs.

For example, with the use of a few helper functions, the following script outputs the bulk of the LaTeX code for Table 3.6:

```
# load a previously stored problem series into memory
PS = load_PS(picklefilename)
for problem in PS:
    global_result = get_global_result(problem)
    SDP_result = get_SDP_result(problem)
    iter0_result = SDP_result.sequence[0]
    print ' & '.join(['',
                      pname_transform(problem.pname),
                      format_perc(iter0_result.get_gap_to(global_result)),
                      format_dec2(iter0_result.solve_time),
                      ''])
```

and the following script generates and saves the graphs used in Figures 3.4 and 3.5:

```
def SDP_gap_iters(problem, SDP_result):
    global_result = get_global_result(problem)
    gap_list = []
    for SDP_result_minor in SDP_result.sequence:
        gap_list.append(SDP_result_minor.get_gap_to(global_result, typ='rel'))
    return gap_list

for i, problem in enumerate(PS1):
    gap_list1 = SDP_gap_iters(problem, get_SDP_result(problem))
    gap_list2 = SDP_gap_iters(problem, get_SDP_result(PS2[i]))
```

```
    gap_list3 = SDP_gap_iters(problem, get_SDP_result(PS3[i]))
    x1 = range(len(gap_list1))
    x2 = range(len(gap_list2))
    x3 = range(len(gap_list3))

    # some calculations to get an appropriate range
    ymin = min(gap_list1 + gap_list2 + gap_list3)
    ymax = max(gap_list1 + gap_list2 + gap_list3)
    delta = abs(ymax - ymin)
    ymin = max(0, ymin-0.3*delta)

    # plotting data
    fig, ax = make_axes_iter(5, ylim=[ymin, ymax + 0.4*delta], figsize=(8, 6))
    ax.plot(x3, gap_list3, 'b-d', lw=1, label='SDPbaselim')
    ax.plot(x2, gap_list2, 'r-o', lw=1, label='SDPheurlim')
    ax.plot(x1, gap_list1, 'g-*', lw=1, label='SDPheur')

    # graph formatting and saving
    ax.set_xticks(range(5))
    plt.legend(loc=2)
    label_ax(ax, 'iteration', 'gap %', title=pname_transform[problem.pname])
    fig.tight_layout()
    savefig2('comparing-methods-{0}'.format(problem.pname))
```

5.1.4 Language printing

We provide functions that can export a given QPCC as either an AMPL or GAMS model, as well as in a 'human readable' format. For example, for the toy problem in the chapter's first code snippet, the constraints can be printed in AMPL-readable format using:

```
from languages import model_printing as mp
ampl_writer = mp.ExpressionWriter(['x', 'y'], typ='AMPL')
ampl_writer.write_eqs(P.A, P.b)
ampl_writer.write_ineqs(P.G, P.h)
ampl_writer.write_comps(P.G, P.h, P.comp)
```

which produces the AMPL code:

```
    eq0:      x + y = 0.5;
    ineq0:      -x <= 0;
    ineq1:      -y <= 0;
    ineq2:      x + y <= 1;
    comp0and1:      -x <= 0   complements   -y <= 0;
```

Complementary constraints can be written in up to three formulations, depending on the language in use. The first formulation is that seen above, which is only supported for AMPL and human

readable format. The second formulation uses a method `write_comps_as_product` which expresses the complementarity constraint as a quadratic equality constraint. Using GAMS format for example, in GAMS format the complementarity constraint is written as

```
comp0and1 ..      (x)*(y) =e= 0;
```

The third formulation writes the complementarities as big M constraints using an auxillary variable `z01`

```
comp0and1a ..      x =l= 10000*z01;
comp0and1b ..      y =l= 10000*(1-z01);
```

where M can be provided manually or will be set to 10000 by default.

The languages module includes wrapper functions which print full models with all necessary declarations, constraints, and run commands, and with support for basic settings such as time limits, initial solutions, etc.

5.1.5 Solve batch manager and PyNEOS

The NEOS server is a fantastic resource which provides free access to all sorts of solvers, including many which would otherwise require paid licenses. We provide an interface to the NEOS server which is based on the one by Dominique Orban (https://www.gerad.ca/ orban/pyneos/). We have extended PyNEOS to handle GAMS and SDPT3 problems as well as AMPL problems, and we provide an interface for solving QPCCs on NEOS using solvers which are compatible with AMPL or GAMS, or for solving SDP problems on NEOS using SDPT3.

To manage series of tests systematically, we provide a problem batch manager, PyNEOS_batch which acts as a collector for batches of NLP solve requests, then uses multithreading to handle sending up to k problems to the NEOS server at a time, retrieving the results, and incorporating them back into their associated problems.

```
PS = load_PS(picklein)

# build basic timestamped options with a few customizations
wsopts = op.get_opts('NLP', {'force_timestamp': cs.get_timestamp(),
                             'NLP_start': 'warmstart',
                             'NLP_reslim': 1800})
globalopts = op.get_opts('NLP', {'force_timestamp': cs.get_timestamp(),
                                 'NLP_start': 'warmstart',
                                 'NLP_reslim': 21600})

# create an empty pyneos batch to manage the solves we will do
```

```
pybatch = PyNEOS_batch(batchfolder)

# print the basic NLP models for the solvers we will use
modfile_guide = prep_modfiles(PS, solvers, pybatch.basefolder)

# load the pybatch problem queue with requests to do local NLP solves
# warmstarted from the candidate point computed with function cand
for i, problem in enumerate(PS):
    SDP_result = get_SDP_result(problem).sequence[0]
    for solver in solvers:
        # Warmstarted local NLP solves
        PS[i] = queue_up_NLP(pybatch, problem, solver, wsopts,
                             modfile_paths=modfile_guide[i],
                             suggested_sol = cand(X))

# solve the problems in the queue and return their results to the problem series
pybatch, PS = solve_and_resolve(pybatch, PS, pickleout)

# now load the queue with requests to do BARON MIP solves warmstarted from
# the KNITRO solves we just did
for i, problem in enumerate(PS):
    NLP_result = get_last_result(problem, solver='KNITRO')
    PS[i] = queue_up_NLP(pybatch, problem, 'BARONMIP', globalopts,
                         modfile_paths=modfile_guide[i],
                         warmstart_res=NLP_result)

# solve the problems in the queue and return their results to the problem series
pybatch, PS = solve_and_resolve(pybatch, PS, pickleout)

# print out a summary of the results the problem series now has
print_summary(PS)
```

5.2 PyQPECgen

We provide a Python implementation of the test problem generator QPECgen, which produces QPEC problems which can be exported as QPCCs. The Python implementation has been developed with the blessing of Houyuan Jian and Danny Ralph, who originally developed QPECgen for Matlab (Jiang and Ralph, 1999).

In PyQPECgen, we organize the different problem types using a hierarchy of object classes. We also extend the generator in two ways:

1. We implement a new problem type which is feasible and bounded even for nonconvex objective functions.

2. For each problem type, we implement a method which exports the problem as a QPCC object as used by PyQPCC.

5.2.1 Introduction

QPECgen is designed to randomly generate feasible QPEC problems according to a problem type and parameters provided by the user. It is notable for giving the user control over many aspects of the problem such as degeneracy, and for the byproducts it produces: a solution which is optimal in the case of a convex objective and feasible otherwise, as well as the values of relevant dual and supplementary variables at this solution. The most general QPEC type generated by QPECgen is the affine variational inequality QPEC (AVI-QPEC), which can be written as

$$\begin{aligned} \min_{x\in\mathbb{R}^n, y\in\mathbb{R}^m, \lambda\in\mathbb{R}^p} \quad & \tfrac{1}{2}\begin{bmatrix} x^T & y^T \end{bmatrix} P \begin{bmatrix} x \\ y \end{bmatrix} + c^T x + d^T y \\ \text{s.t.} \quad & A\begin{bmatrix} x \\ y \end{bmatrix} + a \leq 0 \\ & Dx + Ey + b \leq 0 \\ & \lambda \geq 0 \\ & (Dx + Ey + b)^T \lambda = 0 \\ & Nx + My + q = -E^T \lambda \end{aligned} \tag{AVI-QPEC}$$

effectively modeling the bilevel problem

$$\begin{aligned}
\min_{x\in\mathbb{R}^n, y\in\mathbb{R}^m} \quad & \tfrac{1}{2}\begin{bmatrix} x^T & y^T \end{bmatrix} P \begin{bmatrix} x \\ y \end{bmatrix} + c^T x + d^T y \\
\text{s.t.} \quad & A\begin{bmatrix} x \\ y \end{bmatrix} + a \leq 0 \\
& y \in \operatorname*{argmin}_{y\in\mathbb{R}^m} \; y^T \begin{bmatrix} N & M \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \\
& \qquad \text{s.t.} \quad Dx + Ey + b \leq 0 \;\; ,
\end{aligned} \tag{AVI-BP}$$

The other two main problem types are special cases of (AVI-QPEC). The (BOX-QPEC) has the form

$$\begin{aligned}
\min_{x\in\mathbb{R}^n, y\in\mathbb{R}^m, \lambda\in\mathbb{R}^p} \quad & \tfrac{1}{2}\begin{bmatrix} x^T & y^T \end{bmatrix} P \begin{bmatrix} x \\ y \end{bmatrix} + c^T x + d^T y \\
\text{s.t.} \quad & A\begin{bmatrix} x \\ y \end{bmatrix} + a \leq 0 \\
& 0 \leq y_1 \leq u \\
& 0 \leq y_2 \\
& N_1 x + M_1 y + q_1 = \lambda_{l_1} - \lambda_{u_1} \\
& \lambda_{l_1},\ \lambda_{u_1} \geq 0 \\
& y_1^T \lambda_{l_1} = 0 \\
& (y_1 - u)^T \lambda_{u_1} = 0 \\
& N_2 x + M_2 y + q_2 \geq 0 \\
& y_2^T (N_2 x + M_2 y + q_2) = 0
\end{aligned} \tag{BOX-QPEC}$$

and models a special case of the (AVI-QPEC) in which the lower level problem of the QPEC has the form

$$\begin{aligned}
\min_{y_1\in\mathbb{R}^{m_1}, y_2\in\mathbb{R}^{m_2}} \quad & \tfrac{1}{2}\begin{bmatrix} y_1^T & y_2^T \end{bmatrix} \begin{bmatrix} N_1 & M_1 \\ N_2 & M_2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + q_1^T y_1 + q_2^T y_2 \\
\text{s.t.} \quad & 0 \leq y_1 \leq u \\
& 0 \leq y_2 \;\; .
\end{aligned} \tag{BOX-QPEC-lower}$$

Alternatively, the constraints

$$\begin{aligned} &0 \le y_1 \le u \\ &N_1 x + M_1 y + q_1 = \lambda_{l_1} - \lambda_{u_1} \\ &\lambda_{l_1},\ \lambda_{u_1} \ge 0 \\ &y_1^T \lambda_{l_1} = 0 \\ &(y_1 - u)^T \lambda_{u_1} = 0 \end{aligned} \tag{5.1}$$

can be seen as a 'double complementarity' constraint, which can be programmed in AMPL as

$$0 \le y_1 \le u \qquad \texttt{complements} \qquad N_1 x + M_1 y + q_1 \tag{5.2}$$

or expressed using nonconvex quadratic inequalities without the need for variables $\lambda_{l_1}, \lambda_{u_1}$:

$$\begin{aligned} y_i (N_1 x + M_1 y + q_1)_i &\le 0 \\ (u_i - y_i)(N_1 x + M_1 y + q_1)_i &\ge 0 \ . \end{aligned} \tag{5.3}$$

We refer to (5.1) and (5.3) as the standard complementarity form and quadratic inequality form of the double complementarity constraint, respectively.

The next QPEC type, the (LCP-QPEC), has the form

$$\begin{aligned} \min_{x \in \mathbb{R}^n, y \in \mathbb{R}^m} \quad & \tfrac{1}{2} \begin{bmatrix} x^T & y^T \end{bmatrix} P \begin{bmatrix} x \\ y \end{bmatrix} + c^T x + d^T y \\ \text{s.t.} \quad & A \begin{bmatrix} x \\ y \end{bmatrix} + a \le 0 \\ & y \ge 0 \\ & Nx + My + q \ge 0 \\ & y^T (Nx + My + q) = 0 \ . \end{aligned} \tag{LCP-QPEC}$$

The final two types are very specific special cases of the (LCP-MPEC) which contain no random components and are completely determined by the given size parameters m and n:

$$\begin{aligned} \min_{x \in \mathbb{R}^n, y \in \mathbb{R}^m} \quad & x^T x + y^T y - 2 \sum x + 4 \sum y \\ \text{s.t.} \quad & y \ge x \\ & y \ge 0 \\ & (y - x)^T y = 0 \ , \end{aligned} \tag{GOOD-LCP-QPEC}$$

$$\begin{aligned} \min_{x \in \mathbb{R}^n, y \in \mathbb{R}^m} \quad & x^T x + y^T y + 2 \sum x - 4 \sum y \\ \text{s.t.} \quad & y \ge x \\ & y \ge 0 \\ & (y - x)^T y = 0 \ . \end{aligned} \tag{BAD-LCP-QPEC}$$

5.2.2 Generation options

The problem data, for example $P \in \mathbb{R}^{(n+m)\times(n+m)}, c \in \mathbb{R}^n, d \in \mathbb{R}^m, A \in \mathbb{R}^{l\times(n+m)}, a \in \mathbb{R}^l, D \in \mathbb{R}^{p\times n}, E \in \mathbb{R}^{p\times m}$, $b \in \mathbb{R}^p$, $N \in \mathbb{R}^{m\times n}, M \in \mathbb{R}^{m\times m}, q \in \mathbb{R}^m$ in the case of (AVI-QPEC), are generated according to user given parameters. The PyQPECgen implementation supports the same options as QPECgen, with parameters named as in Table 5.1.

Table 5.1 PyQPECgen parameters

Parameter	Type	Default	Description
qpec_type	int	–	numeric code for problem type, see Fig. 5.2.
n	int	–	number of upper level variables
m	int	–	number of lower level variables
l	int	–	number of upper level constraints
p	int	= m	number of lower level constraints, number of λ variables
cond_P	float	20	condition number of P
scale_P	float	= cond_P	scaling constant of P
convex_f	boolean	True	whether P is generated to be positive semidefinite.
symm_M	boolean	True	whether or not M is symmetric
mono_M	boolean	True	whether or not M is monotone
cond_M	float	10	condition number of the matrix M
scale_M	float	= cond_M	scaling constant of the matrix M
first_deg	int	–	number of degenerate first level constraints
second_deg	int	–	number of degenerate second level constraints
mix_deg	int	0	number of additional degenerate constraints to be allocated among both levels
tol_deg	float	10^{-6}	tolerance in judging degeneracy
rand_seed	–	–	seed value for reproducibility (not yet implemented)
y_upper	boolean	True	whether or not the lower level variables y appear in the upper level constraints

Note: PyQPECgen's 'y_upper' parameter replaces QPECgen's 'constraints' parameter. Additionally, the functionality of rand_seed is not yet implemented.

5.2.3 Python implementation and modular code design

We have chosen to define each problem type as an object class and to organize the object classes in a hierarchy. Compared with generating all problem types with a single script, a class hierarchy has the advantage that code which is unique to one problem type can be isolated to a single class,

making it easier to ensure that it will only be executed in the intended case. Compared with using a separate generation script for each problem type, a class hierarchy has the advantage that code which is common to several problem types can be implemented in the appropriate class(es) in the hierarchy for minimal repetition.

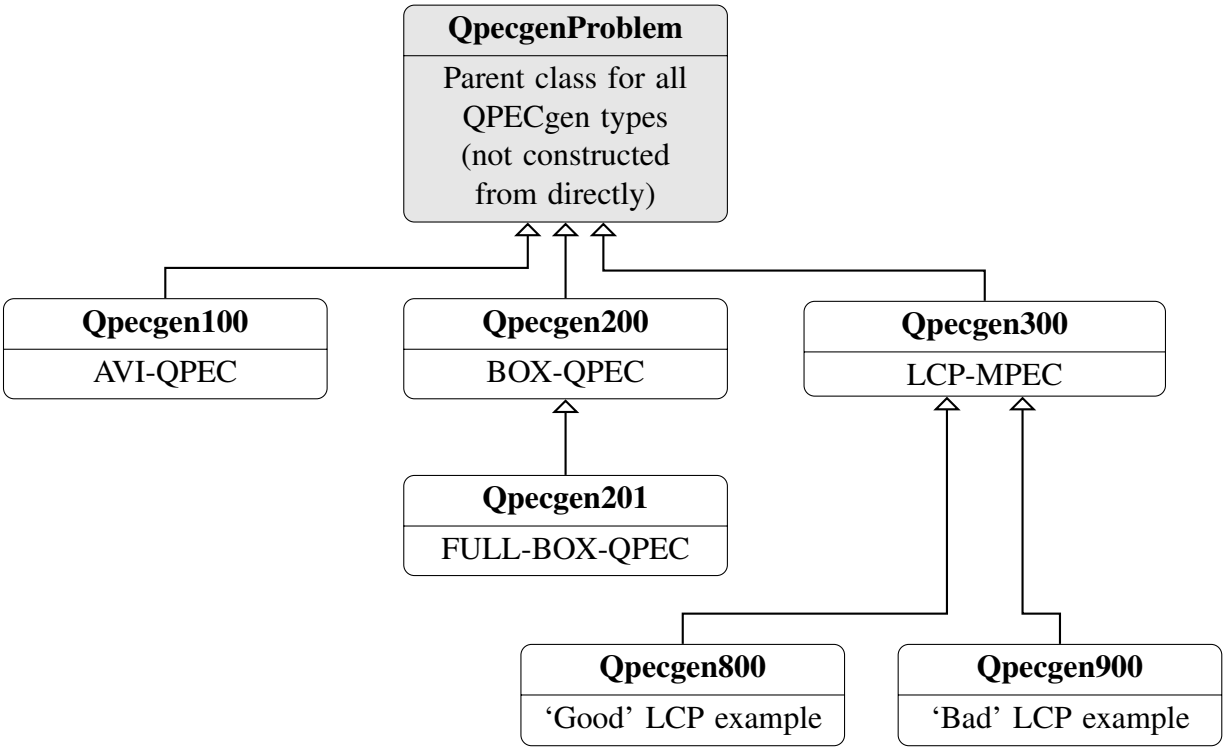

Figure 5.2 Hierarchy of PyQPECgen problem object classes

The class hierarchy we use for PyQPECgen is seen in 5.2. While technically all the QPECgen problem types can be expressed as special cases of the AVI-QPEC, we choose to view the AVI, BOX, and LCP problem types as siblings in the hierarchy because organizing the classes that way makes it easiest to ensure the correct implementation of the BOX and LCP types.

QpecgenProblem is not a problem type which is intended to be constructed directly. Rather, it exists as the superclass of all Qpecgen problem classes in order to manage the properties and methods which are common to all problem types. Specifically, that includes the checking and setting of parameters, as well as the generation of those model components whose generation does not vary with problem type: P, M, N, and A.

Each subclass of QpecgenProblem has additional commands in its initialization method to generate those model components which vary by problem type. Table 5.2 shows which model components are defined for each problem class.

Table 5.2 QPECgen classes and their model components

Problem class	Common components	Class dependent components
Qpecgen100	P, N, M, A	c, d, a, q, D, E, b
Qpecgen200		c, d, a, q, u
Qpecgen201		c, d, a, q, xl, xu, u
Qpecgen300		c, d, a, q
Qpecgen800		c, d, a, q
Qpecgen900		c, d, a, q

5.2.4 New problem type: FULL-BOX-QPEC ('Type 201')

Qpecgen has a parameter 'convex' which controls whether QPECs are generated with convex or general (not necessarily convex) quadratic objective functions. However, for the (AVI-QPEC), (BOX-QPEC), and (LCP-QPEC) problem types, the randomly generated constraints do not necessarily bound the feasible region, and it is commonly the case in practice that problems generated with the option 'convex = False' will be unbounded. To generate test problems with nonconvex objective functions but bounded solutions, we create a bounded variation of the (BOX-QPEC) and call it (FULL-BOX-QPEC):

$$
\begin{aligned}
\min \quad & \tfrac{1}{2}\begin{bmatrix} x^T & y^T \end{bmatrix} P \begin{bmatrix} x \\ y \end{bmatrix} + c^T x + d^T y \\
\text{s.t.} \quad & A \begin{bmatrix} x \\ y \end{bmatrix} \leq -a \\
& 0 \leq x \leq u_x \\
& 0 \leq y \leq u \\
& \lambda_l,\ \lambda_u \geq 0 \\
& y^T \lambda_l = 0 \\
& (y-u)^T \lambda_u = 0 \\
& Nx + My + q = \lambda_l - \lambda_u \quad .
\end{aligned}
\tag{FULL-BOX-QPEC}
$$

Note that it is not necessary to construct constraints explicitly bounding the λ variables. They do not appear in the objective function, so bounding all the other variables is sufficient to ensure that the problem has bounded optimal value. Furthermore, the constraints

$$\begin{aligned} 0 \le y \le u \qquad & \qquad y^T \lambda_l = 0 \\ \lambda_l,\ \lambda_u \ge 0 \qquad & \qquad (y-u)^T \lambda_u = 0 \\ Nx + My + q = \lambda_l - \lambda_u & \end{aligned} \tag{5.4}$$

are the optimality conditions of the QPEC's implicit lower level problem

$$\begin{aligned} \min_{y} \quad & y^T Nx + \tfrac{1}{2} y^T My + q^T y \\ \text{s.t.} \quad & 0 \le y \le u \ , \end{aligned} \tag{FULL-BOX-LOWER}$$

where λ_l, λ_u are the dual variables of the constraints of (FULL-BOX-LOWER). The KKT conditions for (FULL-BOX-LOWER) state that y is optimal for that problem if and only if there exists a finite λ such that the constraints of (5.4) are satisfied. Therefore for the complementarity problem (FULL-BOX-QPEC) there exists a bounded optimal solution which attains the optimal value.

5.2.5 New feature: exporting problem types in the standard QPCC format

For each problem class, we also define a method `return_problem` which constructs a dictionary containing Q, p, r, A, b, G, h which formulate the QPEC as a QPCC in the standard form of (P).

5.2.6 Project status and future direction

This software has been released open source under the MIT License and is publicly available at `http://www.github.com/TrishGillett/pyqpecgen`. While the chief goal of this package is to re-implement the QPECgen generator in Python, and this work is largely complete, members of the public are welcome to give feedback or suggest improvements. Unless otherwise stated, all the features which have been described are presently implemented and stable.

5.3 PySDPT3glue

PySDPT3glue is a software enabling SQLP problems modeled using CVXPY to be solved using the Matlab-specific solver SDPT3, either by accessing a locally installed copy of Matlab with SDPT3 or by submitting the problem to the NEOS server (Czyzyk et al., 1998; Dolan, 2001; Gropp and Moré, 1997). This functionality is primarily of interest to researchers who would like to integrate Matlab's SDPT3 solver into otherwise Python-based methodologies.

CVXPY is a Domain Specific Language (DSL) for modeling convex optimization problems in Python (Diamond and Boyd, 2015). It provides a syntax for building models in an intuitive way and also serves as an interface to a number of Python-accessible solvers, among which MOSEK, CVXOPT, and SCS are capable of handling SQLP problems. PySDPT3glue serves as companion software to CVXPY, allowing SQLP solves to be performed with SDPT3 on Matlab and the results returned to the Python instance which ordered the solve.

5.3.1 Overview of notable functionality

This section describes some of key functions provided by PySDPT3glue.

- `write_cvxpy_to_mat(problem_data, matfile_target)`

  Given CVXOPT problem data such as that exported from a CVXPY problem, build the equivalent problem in Sedumi format (with or without simplification) and save it as a Matlab workspace .mat binary file.

- `msg = matlab_solve(matfile_path, output_target)`

  Given the path to a .mat file containing a Sedumi format problem, solve it with a local Matlab and SDPT3 installation and return the text contents of the output log.

- `msg = neos_solve(matfile_path, output_target)`

  Alternatively, given the path to a .mat file containing a Sedumi format problem, submit the problem to be solved on the NEOS server and retrieve the text contents of the output log.

- `res_dict = make_result_dict(msg)`

  Given the output log of an SDPT3 solve, construct a result dictionary.

These are further wrapped into a single convenience function:

- `res_dict = sdpt3_solve_problem(problem, mode, matfile_target,`

```
output_target)
```

Given a CVXPY problem, a choice of mode as either 'matlab' or 'neos', and paths where the .mat binary file and solver output log should be saved to, the problem is written to a .mat file, solved in the desired way, and a result dictionary is returned.

5.3.2 Problem translation and simplification

A CVXPY problem can be exported in the format used by CVXOPT format provides $c, A, b, G_l, h_l, G_q, h_q, G_s, h_s$ for an SQLP problem having the following form:

$$\begin{array}{lll} min & \bar{c}^T x & \\ \text{s.t.} & \bar{A}x = \bar{b} & \\ & \bar{h_l} - \bar{G_l}x \geq 0 & \\ & \bar{h_{q_i}} - \bar{G_{q_i}}x \in SOC & i = 1, \ldots, K_q \\ & Vec(\bar{h_{s_i}}) - Vec(\bar{G_{s_i}}x) \in \mathbb{S}_+^{k_i} & i = 1, \ldots, K_s \quad . \end{array} \tag{5.5}$$

On the other hand, the SDPT3 solver accepts problems in either Sedumi or SDPA format. We will translate a CVXOPT problem to Sedumi format by first naively rewriting the problem using a large number of supplementary variables and then simplifying the problem by identifying variables and constraints which can be eliminated using simple substitution.

Let n_e and n_i denote the number of linear equality and linear inequality constraints, respectively. Let s_i denote the number of columns of the i^{th} semidefinite programming (SDP) constraint. Let n_x denote the number of variables in the CVXOPT problem. Let s_t denote the total number of elements in all SDP matrix constraints, i.e. $s_t = \sum_{i=1}^{K_s} s_i^2$.

Step 1: Naively rewrite the problem.

Rewrite the problem in the following form:

$$\begin{array}{lll} \min & c^T x & \\ \text{s.t.} & Ax = b & \\ & x = \begin{bmatrix} x_f \\ x_l \\ x_{q_1} \\ \vdots \\ x_{s_1} \\ \vdots \end{bmatrix} & \begin{array}{ll} x_f \in \mathbb{R}^{n_f} & \\ x_l \in \mathbb{R}_+^{n_l} & \\ x_{q_i} \in SOC^{n_{q_i}} & \forall\ i = 1, \ldots, K_q \\ & \\ x_{s_i} \in \mathbb{S}_+^{n_{s_i}} & \forall\ i = 1, \ldots, K_s \quad . \end{array} \end{array} \tag{5.6}$$

This is accomplished naively at first by defining c, A, b as follows:

$$c = \begin{bmatrix} \bar{c} \\ 0_{n_i} \\ 0_{q_t} \\ 0_{s_t} \end{bmatrix}, \qquad A = \begin{bmatrix} \bar{A} & 0_{n_e \times n_i} & 0_{n_e \times q_t} & 0_{n_e \times s_t} \\ \bar{G}_l & I_{n_i} & 0_{n_i \times q_t} & 0_{n_i \times s_t} \\ \bar{G}_q & 0_{q_t \times n_i} & I_{q_t} & 0_{q_t \times s_t} \\ \bar{G}_s & 0_{s_t \times n_i} & 0_{s_t \times q_t} & I_{s_t} \end{bmatrix}, \qquad b = \begin{bmatrix} \bar{b} \\ \bar{h}_l \\ \bar{h}_q \\ \bar{h}_s \end{bmatrix}, \tag{5.7}$$

where x_f be the variables of the original problem, x_l are slack variables created for the linear inequality constraints, x_{q_i} are vectors of slack variables created for the SOC constraints, and x_{s_i} are vectors of slack variables created for the positive semidefinite constraints.

Step 2: Substitute to eliminate use of certain variables.

Eliminate the use of variables by substitution where it can be easily done, and where it is feasible to eliminate the variable completely from the problem. At this time, two types of eliminations are handled:

1. Row k of $Ax = b$ is a constraint of the form $a_{ki}x_i + a_{kj}x_j = b_k$ for nonzero coefficients a_{ki}, a_{kj}, and variable x_i is a part of x_f. Then we can eliminate the use of variable x_i by the following elementary operations:

$$\begin{aligned} A_{:,j} &\leftarrow A_{:,j} - \tfrac{A_{kj}}{A_{ki}} A_{:,i} \\ b &\leftarrow b - \tfrac{b_k}{A_{ki}} A_{:,i} \\ c_j &\leftarrow c_j - \tfrac{A_{kj}}{A_{ki}} c_i \\ d &\leftarrow d + \tfrac{b_k}{A_{ki}} c_i \\ A_{:,i},\ c_i &\leftarrow 0 \ . \end{aligned} \tag{5.8}$$

2. Row k of $Ax = b$ is a constraint of the form $a_{ki}x_i = b_k$ for nonzero coefficient a_{ki}, and variable x_i is a part of either x_f or x_l. In the latter case we also require that $\frac{b_k}{a_{ki}} \geq 0$, because $\frac{b_k}{a_{ki}} < 0$ would indicate the infeasibility of the problem since variables x_l must be in the nonnegative cone. In such a case we leave this infeasible constraint in the problem so that an infeasible result will be passed back from the solver to the user.

$$\begin{aligned} b &\leftarrow b - \tfrac{b_k}{A_{ki}} A_{:,i} \\ d &\leftarrow d + \tfrac{b_k}{A_{ki}} c_i \\ A_{:,i},\ c_i &\leftarrow 0 \ . \end{aligned} \tag{5.9}$$

Step 3: Reduce the variable space.

Reduce the variable space by removing trivial x_f, x_l, x_q variable columns from the problem.

The conditions for removing a variable of each type are as follows:

x_f: A free variable x_i may be removed from the problem if $A_{:,i}$ has no nonzero elements and the objective coefficient c_i is zero.

x_l: A nonnegatively constraint variable x_l may be removed from the problem if $A_{:,i}$ has no nonzero elements and coefficient c_i is nonnegative.

x_q: A variable x_i which is an element in a second order cone constraint may be removed from the problem if it is not the first element of the second order cone, $A_{:,i}$ has no nonzero elements, and the objective function coefficient c_i is nonnegative.

Step 4: Symmetrize the use of PSD matrix variables.

SDPT3 requires that PSD matrix variables be used in a symmetric way, meaning that the variables which correspond to elements (i, j) and (j, i) of the same PSD matrix X_k must have equal coefficients in A and c. This is assured by the following procedure:

$c^\circ = n_f + n_l + \sum_{i=1}^{K_q} s_{q_i}$
for k in $1 \ldots K_s$ **do**
  for i in $1 \ldots s_k$ **do**
    for j in $i + 1 \ldots s_k$ **do**
      $c^U = c^\circ + is_k + j$
      $c^L = c^\circ + js_k + i$
      $A_{:,c^U},\ A_{:,c^L} \leftarrow 0.5(A_{:,c^U} + A_{:,c^L})$
    end for
  end for
  $c^\circ \leftarrow c^\circ + s_k^2$
end for

Step 5: Remove trivial rows of $A^*x = b^*$.

Remove any constraints of $Ax = b$ which have become trivial during the simplification. This is done simply by removing row k of A, b if $A_{k,:}$ has no nonzero elements and $b_k = 0$.

5.3.3 Project status and future direction

This software has been released open source under the MIT License and is publicly available at `http://www.github.com/TrishGillett/pysdpt3glue`. Going forward, members of the public are welcome to give feedback or contribute improvements to the software.

All the features which have been described are presently implemented and stable. A future feature of particular interest is to provide a method for projecting the solution to a simplified problem back to the space of the original problem. Currently, simplification must be disabled if a solution to the original problem is required.

CHAPTER 6 CONCLUSION

6.1 Advancement of knowledge

The research contained in this thesis advances the state of knowledge for semidefinite programming relaxations of QPCCs. We have investigated numerous tightening constraints and made recommendations about which constraint types are likely to be most effective and most practical computationally, particularly the extremely low cost, high benefit aggregated equality constraint. We have proposed an iterative cutting plane method which allows the benefit of the theoretically tight but computationally impractical relaxation (S_{full}) at the cost of adding only a moderate number of constraints. The model we arrive at performs well consistently for the problems examined in this thesis.

We have introduced and explored the notion of a candidate point, examined a number of potential candidate point derivations, and discussed their strengths and weaknesses. The candidate point concept is applicable to SDP relaxations of other problems, and opens the door for semidefinite relaxations to be used other than just for their bounds. We use the candidate point to warmstart local solvers with much success, and show proof of concept for both the warmstarting of the global solver BARON as well as the potential for a primarily SDP-based global branch and bound algorithm.

We have developed a flexible and robust platform for modeling QPCCs and performing analyses and computational experiments. We hope that the PyQPCC, PyQPECgen, and PySDPT3glue packages can help the MPEC research community experiment more freely while being less restricted by borders of languages and solvers.

6.2 Limits and constraints

Each of the test problems considered in this thesis follows one of two particular structures. Unfortunately, there is not a wealth of nonconvex QPLCC test problems in the target problem size range and so we have depended on those problems which can be generated by QPECgen/PyQPECgen.

It is unclear how much instability is a factor in SQLP solvers as a whole as opposed to just an issue with one solver or another. What is clear is that care must be taken with regards to stability when making SDP relaxations of continuous problems whose variables are not necessarily constrained to be small. It is possible that another SQLP solver now or in the future could provide sufficient stability while also having superior times.

Of course, a weakness of the semidefinite relaxation is that for small, simple problems it is exces-

sively computationally expensive compared to other methods. The SDP relaxation also becomes significantly more expensive with each variable that is added to the problem, since the SDP matrix X has size $n + 1 \times n + 1$. On the other hand, a strength of the SDP relaxation is that it is largely unaffected by an increase in the number of complementarity constraints a problem has, where the complexity of some other tools grows exponentially as complementarity constraints are increased.

6.3 Future work

There are a number of simple extensions to (P) which can be modeled well in an SDP context, for example QPLCCs which also have ball constraints or general quadratic constraints. There are also reformulations and preprocessing which were not attempted in this thesis, but which can be done to reduce the complexity of the problem before solving an SDP relaxation. An SDP driven branch and bound implementation for QPLCCs also remains to be attempted.

Regarding the software, there are plans for several improvements. A high priority is the implementation of an interface to MOSEK, a competitive SQLP solver which is compatible with Python. Another desirable feature is to be able to simplify (preprocess) a problem and solve it, and then project the solution back to the space of the original problem; currently the former can be done but not the latter. More generally, there are also plans for a number of changes in order to further reshape PyQPCC into a more general framework that can easily be adapted to the workflow and experiments of other researchers.

REFERENCES

M. S. Andersen, J. Dahl, et L. Vandenberghe, *CVXOPT: A Python package for convex optimization, Version 1.1.6*, 2013.

M. Armbruster, M. Fügenschuh, C. Helmberg, et A. Martin, "A comparative study of linear and semidefinite branch-and-cut methods for solving the minimum graph bisection problem", dans *Integer Programming and Combinatorial Optimization*, série Lecture Notes in Computer Science, A. Lodi, A. Panconesi, et G. Rinaldi, éds. Springer Berlin Heidelberg, 2008, vol. 5035, pp. 112–124. DOI: 10.1007/978-3-540-68891-4. En ligne: http://dx.doi.org/10.1007/978-3-540-68891-4

C. Audet, P. Hansen, B. Jaumard, et G. Savard, "Links between linear bilevel and mixed 0-1 programming problems", *Journal of Optimization Theory and Applications*, vol. 93, no. 2, pp. 273–300, 1997. DOI: 10.1023/A:1022645805569. En ligne: http://dx.doi.org/10.1023/A%3A1022645805569

C. Audet, G. Savard, et W. Zghal, "New branch-and-cut algorithm for bilevel linear programming", *Journal of Optimization Theory and Applications*, vol. 134, no. 2, pp. 353–370, 2007. DOI: 10.1007/s10957-007-9263-4. En ligne: http://dx.doi.org/10.1007/s10957-007-9263-4

C. Audet, P. Hansen, B. Jaumard, et G. Savard, "A symmetrical linear maxmin approach to disjoint bilinear programming", *Mathematical Programming*, vol. 85, no. 3, pp. 573–592, 1999.

C. Audet, J. Haddad, et G. Savard, "Disjunctive cuts for continuous linear bilevel programming", *Optimization Letters*, vol. 1, no. 3, pp. 259–267, 2007.

L. Bai, J. Mitchell, et J.-S. Pang, "On convex quadratic programs with linear complementarity constraints", *Computational Optimization and Applications*, vol. 54, no. 3, pp. 517–554, 2013. DOI: 10.1007/s10589-012-9497-4. En ligne: http://dx.doi.org/10.1007/s10589-012-9497-4

G. Bautista, M. Anjos, et A. Vannelli, "Formulation of oligopolistic competition in ac power networks: An nlp approach", *Power Systems, IEEE Transactions on*, vol. 22, no. 1, pp. 105–115, Feb 2007. DOI: 10.1109/TPWRS.2006.888986

S. Boyd et L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

J. Bracken et J. McGill, “Mathematical programs with optimization problems in the constraints”, *Operations Research*, vol. 21, no. 1, pp. 37–44, 1973. DOI: 10.1287/opre.21.1.37. En ligne: http://dx.doi.org/10.1287/opre.21.1.37

——, “Production and marketing decisions with multiple objectives in a competitive environment”, *Journal of Optimization Theory and Applications*, vol. 24, no. 3, pp. 449–458, 1978. DOI: 10.1007/BF00932888. En ligne: http://dx.doi.org/10.1007/BF00932888

S. Braun et J. Mitchell, “A semidefinite programming heuristic for quadratic programming problems with complementarity constraints”, *Computational Optimization and Applications*, vol. 31, no. 1, pp. 5–29, 2005. DOI: 10.1007/s10589-005-1014-6. En ligne: http://dx.doi.org/10.1007/s10589-005-1014-6

S. Burer et K. Anstreicher, “Second-order-cone constraints for extended trust-region subproblems”, *SIAM Journal on Optimization*, vol. 23, no. 1, pp. 432–451, 2013. DOI: 10.1137/110826862. En ligne: http://dx.doi.org/10.1137/110826862

B. Colson, P. Marcotte, et G. Savard, “Bilevel programming: A survey”, *4OR*, vol. 3, no. 2, pp. 87–107, 2005. DOI: 10.1007/s10288-005-0071-0. En ligne: http://dx.doi.org/10.1007/s10288-005-0071-0

A. Costa et L. Liberti, “Relaxations of multilinear convex envelopes: dual is better than primal”, dans *International Symposium on Experimental Algorithms*. Springer, 2012, pp. 87–98.

Z. Coulibaly et D. Orban, “An l_1 elastic interior-point method for mathematical programs with complementarity constraints”, *SIAM J. on Optimization*, vol. 22, no. 1, pp. 187–211, Mars 2012. DOI: 10.1137/090777232. En ligne: http://dx.doi.org/10.1137/090777232

J. Czyzyk, M. Mesnier, et J. Moré, “The neos server”, *IEEE Journal on Computational Science and Engineering*, vol. 5, no. 3, pp. 68–75, 1998.

S. Dempe, V. Kalashnikov, G. A. Pérez-Valdés, et N. Kalashnykova, *Bilevel Programming Problems: Theory, Algorithms and Applications to Energy Networks*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, ch. Reduction of Bilevel Programming to a Single Level Problem, pp. 41–115. DOI: 10.1007/978-3-662-45827-3_3. En ligne: http://dx.doi.org/10.1007/978-3-662-45827-3_3

S. Diamond et S. Boyd, “Cvxpy: A python-embedded modeling language for convex optimization”, *J Mach Learn Res Mach Learn Open Sour Softw*, 2015.

E. Dolan, “The neos server 4.0 administrative guide”, Mathematics and Computer Science Division, Argonne National Laboratory, Technical Memorandum ANL/MCS-TM-250, 2001.

R. Freund, “15.084j nonlinear programming, spring 2004.” Massachusetts Institute of Technology: MIT OpenCourseWare, 2004. En ligne: http://ocw.mit.edu/courses/sloan-school-of-management/15-084j-nonlinear-programming-spring-2004

S. Gabriel et Y. Smeers, “Complementarity problems in restructured natural gas markets”, dans *Recent Advances in Optimization*, série Lecture Notes in Economics and Mathematical Systems, A. Seeger, éd. Springer Berlin Heidelberg, 2006, vol. 563, pp. 343–373. DOI: 10.1007/3-540-28258-0_21. En ligne: http://dx.doi.org/10.1007/3-540-28258-0_21

W. Gropp et J. Moré, “Optimization environments and the neos server”, dans *Approximation Theory and Optimization*, M. Buhmann et A. Iserles, éds. Cambridge University Press, 1997, pp. 167–182.

P. Hansen, B. Jaumard, et G. Savard, “New branch-and-bound rules for linear bilevel programming”, *SIAM Journal on Scientific and Statistical Computing*, vol. 13, no. 5, pp. 1194–1217, 1992. DOI: 10.1137/0913069. En ligne: http://dx.doi.org/10.1137/0913069

J. Hu, J. Mitchell, J. Pang, K. Bennett, et G. Kunapuli, “On the global solution of linear programs with linear complementarity constraints”, *SIAM Journal on Optimization*, vol. 19, no. 1, pp. 445–471, 2008. DOI: 10.1137/07068463x. En ligne: http://dx.doi.org/10.1137/07068463x

J. Hu, J. Mitchell, J.-S. Pang, et B. Yu, “On linear programs with linear complementarity constraints”, *Journal of Global Optimization*, vol. 53, no. 1, pp. 29–51, 2012. DOI: 10.1007/s10898-010-9644-3. En ligne: http://dx.doi.org/10.1007/s10898-010-9644-3

M. Hu et M. Fukushima, “Variational inequality formulation of a class of multi-leader-follower games”, *Journal of Optimization Theory and Applications*, vol. 151, no. 3, pp. 455–473, 2011. DOI: 10.1007/s10957-011-9901-8. En ligne: http://dx.doi.org/10.1007/s10957-011-9901-8

X. Hu et D. Ralph, “Using epecs to model bilevel games in restructured electricity markets with locational prices”, *Operations Research*, vol. 55, no. 5, pp. 809–827, 2007. DOI: 10.1287/opre.1070.0431. En ligne: http://dx.doi.org/10.1287/opre.1070.0431

Ž. Ivezić, A. J. Connolly, J. T. VanderPlas, et A. Gray, *Statistics, Data Mining, and Machine Learning in Astronomy: A Practical Python Guide for the Analysis of Survey Data*. Princeton University Press, 2014.

H. Jiang et D. Ralph, “Qpecgen, a matlab generator for mathematical programs with quadratic objectives and affine variational inequality constraints”, *Computational Optimization and Applications*, vol. 13, no. 1-3, pp. 25–59, 1999. DOI: 10.1023/A:1008696504163. En ligne: http://dx.doi.org/10.1023/A%3A1008696504163

S. Leyffer, G. López-Calva, et J. Nocedal, “Interior methods for mathematical programs with complementarity constraints”, *SIAM Journal on Optimization*, vol. 17, no. 1, pp. 52–77, 2006. DOI: 10.1137/040621065. En ligne: http://dx.doi.org/10.1137/040621065

Z. Luo, J. Pang, et D. Ralph, *Mathematical Programs with Equilibrium Constraints*. Cambridge University Press, 1996.

O. Mangasarian et S. Fromovitz, “The fritz john necessary optimality conditions in the presence of equality and inequality constraints”, *Journal of Mathematical Analysis and Applications*, vol. 17, no. 1, pp. 37 – 47, 1967. DOI: http://dx.doi.org/10.1016/0022-247X(67)90163-1. En ligne: http://www.sciencedirect.com/science/article/pii/0022247X67901631

J. Nash, “Non-cooperative games”, *Annals of Mathematics*, vol. 54, pp. 286–295, 1951.

Y. Nesterov, A. Nemirovskii, et Y. Ye, *Interior-point polynomial algorithms in convex programming*. SIAM, 1994, vol. 13.

T. Nguyen, M. Tawarmalani, et J.-P. Richard, “Convexification techniques for linear complementarity constraints”, dans *Integer Programming and Combinatoral Optimization*, série Lecture Notes in Computer Science, O. Günlük et G. Woeginger, éds. Springer Berlin Heidelberg, 2011, vol. 6655, pp. 336–348. DOI: 10.1007/978-3-642-20807-2. En ligne: http://dx.doi.org/10.1007/978-3-642-20807-2

G. Rashed et R. Ahsan, “Python in computational science: applications and possibilities”, *International Journal of Computer Applications*, vol. 46, no. 20, pp. 26–30, 2012.

R. T. Rockafellar, “Lagrange multipliers and optimality”, *SIAM Review*, vol. 35, no. 2, pp. 183–238, 1993. DOI: 10.1137/1035044. En ligne: http://dx.doi.org/10.1137/1035044

D. Ševčovič et M. Trnovská, “Solution to the inverse wulff problem by means of the enhanced semidefinite relaxation method”, *Journal of Inverse and Ill-Posed Problems*, 2014.

H. D. Sherali et W. P. Adams, *Reformulation-Linearization Techniques for Discrete Optimization Problems*. Boston, MA: Springer US, 1999, pp. 479–532. DOI: 10.1007/978-1-4613-0303-9_7. En ligne: http://dx.doi.org/10.1007/978-1-4613-0303-9_7

S. Siddiqui et S. Gabriel, “An sos1-based approach for solving mpecs with a natural gas market application”, *Networks and Spatial Economics*, vol. 13, no. 2, pp. 205–227, 2013. DOI: 10.1007/s11067-012-9178-y. En ligne: http://dx.doi.org/10.1007/s11067-012-9178-y

H. Stackelberg, *The Theory of Market Economy*. Oxford University Press, Oxford, UK., 1952.

G. Stewart, “On the early history of the singular value decomposition”, *SIAM Review*, vol. 35, no. 4, pp. 551–566, 1993. DOI: 10.1137/1035134. En ligne: http://dx.doi.org/10.1137/1035134

M. Tawarmalani et N. V. Sahinidis, *Convexification and global optimization in continuous and mixed-integer nonlinear programming: theory, algorithms, software, and applications*. Springer, 2002, vol. 65.

K. C. Toh, R. H. Tütüncü, et M. J. Todd, *On the implementation and usage of SDPT3 – a Matlab software package for semidefinite-quadratic-linear programming, version 4.0*, 2006.

R. H. Tütüncü, K. C. Toh, et M. J. Todd, *SDPT3 – a Matlab software package for semidefinite-quadratic-linear programming, version 3.0*, 2001.

L. Vandenberghe, *The CVXOPT linear and Quadratic cone program solvers*, 2010. En ligne: http://cvxopt.org/documentation/coneprog.pdf

L. Vicente, G. Savard, et J. JÃºdice, “Descent approaches for quadratic bilevel programming”, *Journal of Optimization Theory and Applications*, vol. 81, no. 2, pp. 379–399, 1994. DOI: 10.1007/BF02191670. En ligne: http://dx.doi.org/10.1007/BF02191670

J. Vielma et G. Nemhauser, “Modeling disjunctive constraints with a logarithmic number of binary variables and constraints”, *Mathematical Programming*, vol. 128, no. 1-2, pp. 49–72, 2011. DOI: 10.1007/s10107-009-0295-4. En ligne: http://dx.doi.org/10.1007/s10107-009-0295-4

Y. Ye et S. Zhang, “New results on quadratic minimization”, *SIAM Journal on Optimization*, vol. 14, no. 1, pp. 245–267, 2003. DOI: 10.1137/S105262340139001X. En ligne: http://dx.doi.org/10.1137/S105262340139001X

APPENDIX A DERIVING CANDIDATE POINTS FROM THE SOLUTION TO A HEURISTIC SDP MODEL

Tables A.1 - A.6 show the full results of the candidate point evaluation seen in section 4.4.2.

Table A.1 Evaluating candidate points derived from the solution to (S_{heur}). (B20 series)

	SDP sol. X^*		candidate point $\hat{x}$			
Name	$gap_S(X^*)$ %	$R(X^*)$	type	$gap_c(\hat{x})$	$viol_c(\hat{x})$	$dist_c(\hat{x})$
B20n0	2.46%	0.19	LP	2.46%	0.00	16.40%
			SP	2.46%	5.39	46.66%
			R1	2.46%	0.02	16.39%
			AR1	2.46%	0.02	16.39%
B20n1	0.00%	0.04	LP	0.10%	0.00	1.95%
			SP	0.11%	0.03	20.41%
			R1	0.10%	0.00	1.95%
			AR1	0.10%	0.00	1.95%
B20n2	0.20%	0.58	LP	0.21%	0.00	11.49%
			SP	0.19%	8.10	112.25%
			R1	0.21%	0.02	11.40%
			AR1	0.21%	0.02	11.40%
B20n3	2.61%	0.04	LP	2.61%	0.00	15.19%
			SP	2.60%	0.67	24.47%
			R1	2.61%	0.00	15.19%
			AR1	2.61%	0.00	15.19%
B20n4	0.09%	0.05	LP	0.09%	0.00	6.71%
			SP	0.09%	2.13	22.82%
			R1	0.09%	0.01	6.72%
			AR1	0.09%	0.01	6.72%
B20n5	38.23%	0.09	LP	38.23%	0.00	62.57%
			SP	38.23%	3.60	64.04%
			R1	38.23%	0.03	62.57%
			AR1	38.23%	0.03	62.56%

Table A.2 Evaluating candidate points derived from the solution to (S_{heur}). (F20 series)

	SDP sol. X^*		candidate point $\hat{x}$			
Name	$gap_S(X^*)$ %	$R(X^*)$	type	$gap_c(\hat{x})$	$viol_c(\hat{x})$	$dist_c(\hat{x})$
F20n0	0.00%	0.19	LP	0.00%	0.00	3.35%
			SP	0.00%	3.53	45.87%
			R1	0.00%	0.02	3.30%
			AR1	0.00%	0.02	3.30%
F20n1	0.37%	0.28	LP	0.37%	0.00	9.41%
			SP	0.37%	0.90	56.15%
			R1	0.38%	0.00	9.52%
			AR1	0.37%	0.00	9.52%
F20n2	0.00%	0.17	LP	0.00%	0.00	5.96%
			SP	0.00%	6.10	39.11%
			R1	0.00%	0.03	5.81%
			AR1	0.00%	0.03	5.81%
F20n3	0.00%	0.16	LP	0.00%	0.00	15.49%
			SP	0.01%	2.29	45.98%
			R1	0.01%	0.01	16.42%
			AR1	0.00%	0.01	16.46%
F20n4	0.00%	0.16	LP	0.00%	0.00	14.74%
			SP	0.00%	1.03	46.45%
			R1	0.01%	0.01	15.85%
			AR1	0.00%	0.00	15.90%
F20n5	0.35%	0.21	LP	0.35%	0.00	5.68%
			SP	0.35%	1.84	50.70%
			R1	0.35%	0.00	5.62%
			AR1	0.35%	0.00	5.62%

Table A.3 Evaluating candidate points derived from the solution to (S_{heur}). (B50 series)

	SDP sol. X^*		candidate point $\hat{x}$			
Name	$gap_S(X^*)$ %	$R(X^*)$	type	$gap_c(\hat{x})$	$viol_c(\hat{x})$	$dist_c(\hat{x})$
B50n0	0.05%	0.14	LP	0.05%	0.00	9.28%
			SP	0.05%	1.26	38.48%
			R1	0.05%	0.00	9.28%
			AR1	0.05%	0.00	9.28%
B50n1	2.72%	0.23	LP	2.72%	0.00	24.17%
			SP	2.71%	7.00	56.25%
			R1	2.72%	0.00	24.17%
			AR1	2.72%	0.00	24.17%
B50n2	4.76%	0.25	LP	4.76%	0.00	35.41%
			SP	4.76%	6.76	51.29%
			R1	4.76%	0.02	35.41%
			AR1	4.76%	0.02	35.41%
B50n3	0.14%	0.21	LP	0.14%	0.00	13.53%
			SP	0.14%	6.41	45.95%
			R1	0.14%	0.02	13.48%
			AR1	0.14%	0.02	13.48%
B50n4	0.88%	0.87	LP	0.96%	0.02	37.67%
			SP	0.90%	48.31	272.53%
			R1	94.48%	1.09	95.30%
			AR1	0.97%	87.35	609.70%
B50n5	1.98%	0.16	LP	1.98%	0.00	16.71%
			SP	1.98%	3.17	43.35%
			R1	1.98%	0.01	16.74%
			AR1	1.98%	0.01	16.74%

Table A.4 Evaluating candidate points derived from the solution to (S_{heur}). (F50 series)

	SDP sol. X^*		candidate point $\hat{x}$			
Name	$gap_S(X^*)$ %	$R(X^*)$	type	$gap_c(\hat{x})$	$viol_c(\hat{x})$	$dist_c(\hat{x})$
F50n0	44.47%	0.32	LP	44.48%	0.00	64.78%
			SP	44.45%	13.14	77.55%
			R1	44.48%	0.02	64.77%
			AR1	44.48%	0.02	64.77%
F50n1	6.58%	0.29	LP	6.58%	0.00	33.58%
			SP	6.58%	2.12	68.20%
			R1	6.58%	0.01	33.59%
			AR1	6.58%	0.01	33.60%
F50n2	4.12%	0.55	LP	4.12%	0.00	43.04%
			SP	4.11%	8.72	118.10%
			R1	4.01%	0.06	46.11%
			AR1	4.12%	0.05	46.71%
F50n3	0.00%	0.16	LP	0.00%	0.00	2.28%
			SP	0.00%	9.25	41.91%
			R1	0.00%	0.01	2.28%
			AR1	0.00%	0.01	2.28%
F50n4	0.00%	0.24	LP	0.00%	0.00	24.45%
			SP	0.00%	4.23	58.65%
			R1	0.01%	0.02	25.32%
			AR1	0.00%	0.02	25.40%
F50n5	0.09%	0.25	LP	0.09%	0.00	34.29%
			SP	0.08%	1.71	58.28%
			R1	0.08%	0.00	34.59%
			AR1	0.09%	0.00	34.61%

Table A.5 Evaluating candidate points derived from the solution to (S_{heur}). (B100 series)

	SDP sol. X^*		candidate point $\hat{x}$			
Name	$gap_S(X^*)$ %	$R(X^*)$	type	$gap_c(\hat{x})$	$viol_c(\hat{x})$	$dist_c(\hat{x})$
B100n0	0.48%	0.05	LP	0.48%	0.00	20.51%
			SP	0.48%	0.38	28.85%
			R1	0.48%	0.00	20.51%
			AR1	0.48%	0.00	20.51%
B100n1	0.28%	0.14	LP	0.28%	0.00	24.29%
			SP	0.28%	4.10	40.80%
			R1	0.28%	0.01	24.28%
			AR1	0.28%	0.01	24.28%
B100n2	0.00%	0.09	LP	0.00%	0.00	8.14%
			SP	0.00%	0.10	31.79%
			R1	0.00%	0.00	8.14%
			AR1	0.00%	0.00	8.14%
B100n3	0.09%	0.27	LP	0.09%	0.00	5.83%
			SP	0.09%	8.20	58.83%
			R1	0.09%	0.01	5.84%
			AR1	0.09%	0.01	5.84%
B100n4	0.10%	0.19	LP	0.10%	0.00	10.39%
			SP	0.10%	4.77	45.08%
			R1	0.10%	0.00	10.37%
			AR1	0.10%	0.00	10.37%
B100n5	0.59%	0.24	LP	0.59%	0.00	29.96%
			SP	0.59%	3.75	56.55%
			R1	0.59%	0.00	29.96%
			AR1	0.59%	0.00	29.96%

Table A.6 Evaluating candidate points derived from the solution to (S_{heur}). (F100 series)

	SDP sol. X^*		candidate point $\hat{x}$			
Name	$gap_S(X^*)$ %	$R(X^*)$	type	$gap_c(\hat{x})$	$viol_c(\hat{x})$	$dist_c(\hat{x})$
F100n0	0.13%	0.22	LP	0.13%	0.00	14.71%
			SP	0.13%	5.64	53.51%
			R1	0.12%	0.00	14.89%
			AR1	0.13%	0.00	14.89%
F100n1	0.84%	0.30	LP	0.84%	0.00	17.01%
			SP	0.84%	6.96	62.38%
			R1	0.84%	0.01	17.11%
			AR1	0.84%	0.01	17.11%
F100n2	0.31%	0.28	LP	0.31%	0.00	26.25%
			SP	0.31%	4.29	63.93%
			R1	0.31%	0.00	26.45%
			AR1	0.31%	0.00	26.46%
F100n3	0.11%	0.20	LP	0.11%	0.00	15.28%
			SP	0.11%	4.72	47.13%
			R1	0.11%	0.01	15.39%
			AR1	0.11%	0.01	15.39%
F100n4	0.06%	0.21	LP	0.06%	0.00	10.52%
			SP	0.05%	5.39	50.60%
			R1	0.06%	0.01	10.64%
			AR1	0.06%	0.01	10.64%
F100n5	0.41%	0.54	LP	0.42%	0.00	31.07%
			SP	0.41%	7.73	112.61%
			R1	0.38%	0.01	32.57%
			AR1	0.42%	0.00	32.76%

APPENDIX B DERIVING CANDIDATE POINTS AFTER AN ITERATIVE SDP PROCESS

Tables B.1 through B.6 parallel the candidate point evaluation seen in section 4.4.2 and Appendix A, but where X^* is the SDP solution from the iterative method after 5 outer iterations or at termination, whichever occurs first.

Table B.1 Evaluating candidate points derived from the last iteration of the $SDP_{heurlim}$ iterative solves. (B20 series)

	SDP sol. X^*		candidate point $\hat{x}$			
Name	$gap_S(X^*)$ %	$R(X^*)$	type	$gap_c(\hat{x})$	$viol_c(\hat{x})$	$dist_c(\hat{x})$
B20n0	2.46%	0.32	LP	2.46%	0.00	16.40%
			SP	2.46%	7.37	67.49%
			R1	2.46%	0.17	16.36%
			AR1	2.46%	0.17	16.37%
B20n1	0.00%	0.04	LP	0.10%	0.00	1.95%
			SP	0.11%	0.03	20.41%
			R1	0.10%	0.00	1.95%
			AR1	0.10%	0.00	1.95%
B20n2	0.20%	0.45	LP	0.20%	0.00	11.84%
			SP	0.19%	5.43	87.30%
			R1	0.20%	0.00	11.84%
			AR1	0.20%	0.00	11.84%
B20n3	2.61%	0.04	LP	2.61%	0.00	15.19%
			SP	2.60%	0.67	24.47%
			R1	2.61%	0.00	15.19%
			AR1	2.61%	0.00	15.19%
B20n4	0.09%	0.05	LP	0.09%	0.00	6.71%
			SP	0.09%	2.13	22.82%
			R1	0.09%	0.01	6.72%
			AR1	0.09%	0.01	6.72%
B20n5	38.23%	0.37	LP	38.23%	0.00	62.57%
			SP	38.23%	8.43	73.11%
			R1	38.23%	0.31	62.54%
			AR1	38.23%	0.31	62.51%

Table B.2 Evaluating candidate points derived from the last iteration of the $SDP_{heurlim}$ iterative solves. (F20 series)

	SDP sol. X^*		candidate point $\hat{x}$			
Name	$gap_S(X^*)$ %	$R(X^*)$	type	$gap_c(\hat{x})$	$viol_c(\hat{x})$	$dist_c(\hat{x})$
F20n0	0.00%	0.33	LP	0.00%	0.00	3.37%
			SP	0.00%	5.26	66.55%
			R1	0.00%	0.09	3.11%
			AR1	0.00%	0.09	3.11%
F20n1	0.37%	0.42	LP	0.37%	0.00	9.42%
			SP	0.37%	1.16	79.09%
			R1	0.40%	0.02	10.30%
			AR1	0.37%	0.02	10.34%
F20n2	0.00%	0.42	LP	0.00%	0.00	5.98%
			SP	0.00%	11.37	77.62%
			R1	0.01%	0.53	5.41%
			AR1	0.00%	0.53	5.39%
F20n3	0.00%	0.39	LP	0.00%	0.00	13.95%
			SP	0.01%	4.49	80.47%
			R1	0.03%	0.23	16.23%
			AR1	0.00%	0.23	16.34%
F20n4	0.00%	0.27	LP	0.00%	0.00	15.78%
			SP	0.00%	1.68	63.95%
			R1	0.03%	0.07	17.99%
			AR1	0.00%	0.07	18.11%
F20n5	0.35%	0.58	LP	0.35%	0.00	5.68%
			SP	0.35%	3.88	116.50%
			R1	0.28%	0.25	8.17%
			AR1	0.35%	0.25	8.31%

Table B.3 Evaluating candidate points derived from the last iteration of the $SDP_{heurlim}$ iterative solves. (B50 series)

	SDP sol. X^*		candidate point $\hat{x}$			
Name	$gap_S(X^*)$ %	$R(X^*)$	type	$gap_c(\hat{x})$	$viol_c(\hat{x})$	$dist_c(\hat{x})$
B50n0	0.05%	0.23	LP	0.05%	0.00	9.28%
			SP	0.05%	1.95	53.13%
			R1	0.05%	0.00	9.28%
			AR1	0.05%	0.00	9.28%
B50n1	2.72%	0.36	LP	2.72%	0.00	24.17%
			SP	2.71%	9.54	74.21%
			R1	2.72%	0.01	24.16%
			AR1	2.72%	0.01	24.16%
B50n2	4.76%	0.52	LP	4.76%	0.00	35.43%
			SP	4.76%	11.51	90.37%
			R1	4.75%	0.33	34.62%
			AR1	4.76%	0.33	34.63%
B50n3	0.14%	0.48	LP	0.14%	0.00	13.54%
			SP	0.14%	11.83	88.55%
			R1	0.14%	0.41	12.89%
			AR1	0.14%	0.41	12.89%
B50n4	0.84%	0.49	LP	0.89%	0.01	37.63%
			SP	0.96%	79.56	442.17%
			R1	99.97%	0.99	99.55%
			AR1	1.02%	4642.21	26470.03%
B50n5	1.98%	0.45	LP	1.98%	0.00	16.71%
			SP	1.98%	6.63	87.74%
			R1	1.98%	0.10	16.73%
			AR1	1.98%	0.10	16.73%

Table B.4 Evaluating candidate points derived from the last iteration of the $SDP_{heurlim}$ iterative solves. (F50 series)

	SDP sol. X^*		candidate point $\hat{x}$			
Name	$gap_S(X^*)$ %	$R(X^*)$	type	$gap_c(\hat{x})$	$viol_c(\hat{x})$	$dist_c(\hat{x})$
F50n0	44.47%	0.58	LP	44.48%	0.00	64.82%
			SP	44.45%	22.03	109.74%
			R1	44.47%	0.39	64.32%
			AR1	44.48%	0.39	64.33%
F50n1	6.58%	0.58	LP	6.58%	0.00	33.59%
			SP	6.58%	4.29	119.86%
			R1	6.58%	0.11	33.41%
			AR1	6.58%	0.11	33.45%
F50n2	4.12%	0.46	LP	4.12%	0.00	33.49%
			SP	4.12%	6.53	90.90%
			R1	4.11%	0.10	33.63%
			AR1	4.12%	0.10	33.68%
F50n3	0.00%	0.40	LP	0.00%	0.00	2.29%
			SP	0.00%	16.84	78.64%
			R1	0.00%	0.18	2.21%
			AR1	0.00%	0.18	2.21%
F50n4	0.00%	0.45	LP	0.00%	0.00	22.71%
			SP	0.00%	7.24	90.89%
			R1	0.01%	0.21	24.47%
			AR1	0.00%	0.20	24.68%
F50n5	0.09%	0.40	LP	0.09%	0.00	34.46%
			SP	0.08%	2.67	77.26%
			R1	0.07%	0.11	34.51%
			AR1	0.09%	0.11	34.55%

Table B.5 Evaluating candidate points derived from the last iteration of the $SDP_{heurlim}$ iterative solves. (B100 series)

	SDP sol. X^*		candidate point $\hat{x}$			
Name	$gap_S(X^*)$ %	$R(X^*)$	type	$gap_c(\hat{x})$	$viol_c(\hat{x})$	$dist_c(\hat{x})$
B100n0	0.48%	0.23	LP	0.48%	0.00	20.52%
			SP	0.48%	0.87	55.07%
			R1	0.48%	0.00	20.52%
			AR1	0.48%	0.00	20.52%
B100n1	0.28%	0.37	LP	0.28%	0.00	24.29%
			SP	0.28%	8.12	73.48%
			R1	0.28%	0.13	24.23%
			AR1	0.28%	0.13	24.22%
B100n2	0.00%	0.27	LP	0.00%	0.00	8.14%
			SP	0.00%	0.20	61.31%
			R1	0.00%	0.00	8.14%
			AR1	0.00%	0.00	8.14%
B100n3	0.09%	0.51	LP	0.09%	0.00	5.85%
			SP	0.09%	12.87	100.52%
			R1	0.09%	0.23	6.14%
			AR1	0.09%	0.23	6.15%
B100n4	0.10%	0.46	LP	0.10%	0.00	10.39%
			SP	0.10%	8.35	86.60%
			R1	0.10%	0.12	10.05%
			AR1	0.10%	0.12	10.05%
B100n5	0.59%	0.49	LP	0.59%	0.00	29.97%
			SP	0.59%	6.81	93.40%
			R1	0.59%	0.01	29.93%
			AR1	0.59%	0.01	29.93%

Table B.6 Evaluating candidate points derived from the last iteration of the $SDP_{heurlim}$ iterative solves. (F100 series)

	SDP sol. X^*		candidate point $\hat{x}$			
Name	$gap_S(X^*)$ %	$R(X^*)$	type	$gap_c(\hat{x})$	$viol_c(\hat{x})$	$dist_c(\hat{x})$
F100n0	0.13%	0.36	LP	0.13%	0.00	14.60%
			SP	0.13%	7.75	73.95%
			R1	0.12%	0.04	14.85%
			AR1	0.13%	0.04	14.86%
F100n1	0.84%	0.53	LP	0.84%	0.00	16.78%
			SP	0.83%	11.11	101.92%
			R1	0.84%	0.14	16.89%
			AR1	0.84%	0.14	16.89%
F100n2	0.31%	0.51	LP	0.31%	0.00	25.65%
			SP	0.31%	6.96	100.41%
			R1	0.31%	0.16	26.14%
			AR1	0.31%	0.16	26.19%
F100n3	0.11%	0.44	LP	0.11%	0.00	14.91%
			SP	0.11%	8.06	84.40%
			R1	0.11%	0.19	15.09%
			AR1	0.11%	0.19	15.10%
F100n4	0.06%	0.44	LP	0.06%	0.00	10.97%
			SP	0.05%	9.04	87.00%
			R1	0.06%	0.11	11.77%
			AR1	0.06%	0.11	11.79%
F100n5	0.41%	0.60	LP	0.41%	0.00	21.10%
			SP	0.41%	9.11	122.33%
			R1	0.34%	0.44	22.81%
			AR1	0.41%	0.44	22.94%